

Code EE1L21

# Manual

Project Smart Robot Challenge

Ing. B. Jacobs, Ing. X. van Rijnsoever, A.M.J. Slats, Dr. J. Hoekstra, Dr. ir. A.J. van Genderen  
Dr. S. Izadkhast, Dr. ir. M. Pertijs, Dr. M. Bartek, Dr. M. Spirito  
Ing. B.M. Verdoes, Ing. T.M. de Rijk, Ing. H.E.J. Bosma

# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Learning goals . . . . .	4
1.2	EPO-2 course structure . . . . .	4
1.3	Working method for the EPO-2 project . . . . .	7
1.4	Final mark . . . . .	9
1.5	Deadlines of deliverables . . . . .	11
1.6	Project organization . . . . .	11
<b>2</b>	<b>Getting to know the Robot</b>	<b>12</b>
2.1	Introduction . . . . .	12
2.2	Overview of the robot . . . . .	12
2.3	Robot hardware: details and source files . . . . .	13
2.4	Robot Software: details and source files . . . . .	19
<b>3</b>	<b>Operating the robot and measuring with the robot</b>	<b>20</b>
3.1	Introduction . . . . .	20
3.2	The battery . . . . .	21
3.3	The optical sensors . . . . .	22
3.4	The servomotors . . . . .	23
3.5	Conclusion . . . . .	23
<b>II</b>	<b>Project Smart Robot Challenge</b>	<b>24</b>
<b>4</b>	<b>Maze router</b>	<b>25</b>
4.1	Introduction . . . . .	25
4.2	Assignment . . . . .	25
4.3	Algorithm . . . . .	26
4.4	Datastructure . . . . .	28
4.5	Development . . . . .	29
<b>5</b>	<b>Wireless Communication between PC and Robot</b>	<b>30</b>
5.1	Introduction . . . . .	30
5.2	The UART Protocol . . . . .	31
5.3	The Robot-XBee link (UART) . . . . .	33
5.4	The XBee-XBee link (Wireless) . . . . .	35
5.5	The XBee-PC link (Windows serial) . . . . .	41

<b>6</b>	<b>Mine Detector Design</b>	<b>43</b>
6.1	Working Principle . . . . .	43
6.2	Design constraints . . . . .	43
6.3	Design Tips . . . . .	44
<b>7</b>	<b>Advanced FSM</b>	<b>45</b>
7.1	Advantages and disadvantages . . . . .	45
7.2	More thoughts on the advanced FSM for the robot . . . . .	46
	<b>Bibliography</b>	<b>48</b>
<b>A</b>	<b>Tutorial ISE</b>	<b>50</b>
A.1	Introduction . . . . .	50
A.2	Hardware: het Xilinx Spartan3E FPGA development board . . . . .	50
A.3	Software: Xilinx ISE . . . . .	52
<b>B</b>	<b>UART</b>	<b>59</b>
B.1	User constraint file uart.ucf . . . . .	59
B.2	Template C program to interface with Windows COM port . . . . .	60
<b>C</b>	<b>Overview of the FPGA pins</b>	<b>63</b>
C.1	7-Segment displays . . . . .	64
C.2	Leds . . . . .	64
C.3	Slide Switches . . . . .	64
C.4	Push buttons . . . . .	65
C.5	Oscillator . . . . .	65
C.6	Full list of connections . . . . .	65

**Part I**

**Introduction**

# Chapter 1

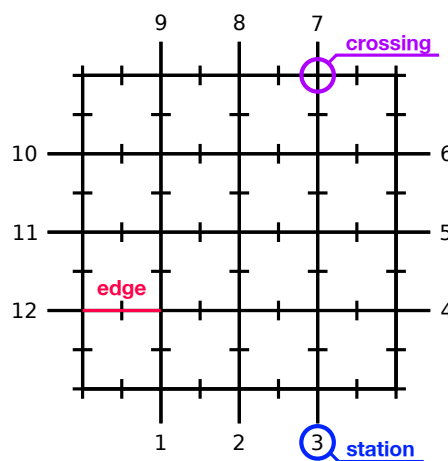
## Introduction

Welcome to the practicum of the second semester, first year. In the third quarter, you followed the EE1C31 and EE1D21 'Course labs'. In the fourth quarter you will carry out the 'EPO-2 project: Smart Robot Challenge'.

### About the Project

During the course *Digital Systems B* you had to write a VHDL description that allowed a robot to track a black line on a white surface. Three infra-red sensors were used to enable you to detect this black line, and two servo motors allowed the robot to adjust its course. During *EPO-2: Smart Robot Challenge*, we will use the same robot, but your task will be more complex than getting it to follow a line.

The goal of the project is to expand the capabilities of the line follower robot such that it can pass three challenges called challenge A, B and C. As opposed to a single-line map, all challenges take place on the map shown in Fig. 1.1. We sometimes refer to this map as "the maze".



**Figure 1.1:** Top view of the map ("the maze") used for challenges A, B and C including terms used to describe properties of the map topology.

Here is a brief overview of each challenge:

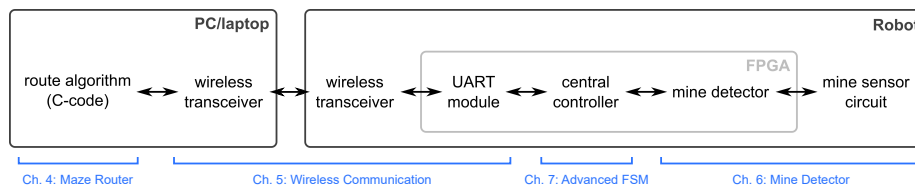
- A** For challenge A the robot is given a sequence of stations that it needs to visit in order. An example could be: start at 1, visit 3 and then 6, and finish at 10.
- B** Challenge B is the same as challenge A, but on the midpoint of some of the maze's edges there will be metal disks that need to be avoided. We call these disks "mines". There will be no mines at stations or crossings.
- C** During challenge C there will be a number of mines on the midpoints of edges of the maze too. The robot is tasked to find all of them. In phase 2, a new mine is placed in the maze, and the robot needs to find that mine too.

At the end of the project there will be a symposium where groups can compete for who can finish the *most challenges*, and who finishes them in the *shortest time*. The specific rules for the symposium will be uploaded to Brightspace.

## About the Manual

This manual is divided into two sections. Section I describes the procedures and the format of the project. It introduces the project, and provides theory on how the robot works. Section II provides information about the design of your system.

Fig. 1.2 gives an overview of the system that you will design with your group during this project. To navigate the maze, you are required to write a C-program that runs on a PC or a laptop that sends commands to and receives data from the robot. To detect mines, you will need to design a mine sensing circuit. And finally you will need to upgrade the VHDL description of your line follower to interface with these new modules, and to enable the robot to traverse the maze. The relevant information and instructions for all parts of this system can be found in the chapters as indicated in the Figure.



**Figure 1.2:** Bird's eye overview of relevant subsystems of the system to be designed for EPO-2 with references to the related chapters in this manual.

Each chapter of this manual contains a brief overview of the learning objectives of that chapter, followed by a theory and/or set of experiments on that subject. To be well prepared for the activities of each session, **it is strongly advised that you read the material of the specific week in advance**. The sessions will be much more interesting and fun if you know what you are doing!

In the manual, there are a number of text boxes showing with different icons. These boxes contain additional information that is not directly necessary for the labs, but it is useful to know.

**Attention**

The frame with this icon warns on common errors and problems.

**Suggestion**

The frame with this icon provides a tip that can be useful in addressing a problem of the assignment.

**Background information**

The frame with this icon provides background information about, for example practical applications, or additional functionalities that might not be required for the exercise, but might be applicable to other practicums.

## 1.1 Learning goals

Below are the learning objectives of the EPO-2 project. These learning goals have been used as a guide to define the content of this practicum and are used to align and guide the final assessment. Students can use them as the goals to work towards and as a continuous check whether the subjects and assignments have been sufficiently understood.

Upon successful completion of EPO2 project, the student will be able to:

- Apply a project-based approach to design an electronic system in a project team;
- Methodologically design a technical solution to a relevant problem within given boundary conditions;
- Integrate and apply, in a design project, the knowledge and skills acquired during the lecture classes of the first year;
- Actively apply academic skills in order to identify and acquire missing information and knowledge, systematically analyze problems, motivate choices, critically interpret results;
- Document own project-related work and write a well-organized technical report.

## 1.2 EPO-2 course structure

Figure 1.3 presents the timeline of EPO-2 in 2023. The kick-off of EPO-2 project will take place on Monday 24th April in Ampere Hall at 12:45 pm.

		Week 4.1		Week 4.2			Week 4.3			Week 4.4	
		Mon 24th Apr	Tue 25th Apr	Mon 1st May	Tue 2nd May	Thu 4th May	Mon 8th May	Tue 9th May	Thu 11th May	Mon 15th May	Tue 16th May
Morning (8:45-12:30)		–	Gr B+C	–	Gr B+C	Gr A	–	Gr B+C	Gr A	–	Gr B+C
Afternoon (13:45 -17:30)		Gr A	–	Gr A	–	Gr B+C	Gr A	–	Gr B+C	Gr A	–

		Week 4.5	Week 4.6	Week 4.7			Week 4.8			Week 4.9	
			Thu 1st Jun	Mon 5th Jun	Tue 6th Jun	Thu 8th Jun	Mon 12th Jun	Tue 13th Jun	Thu 15th Jun	Mon 19th Jun	Tue 20th Jun
Morning (8:45-12:30)			Gr A	–	Gr B+C	Gr A	–	Gr B+C (Testing challenge C)	Symposium Challenge A	Oral exams	
Afternoon (13:45 -17:30)			Gr B+C	Gr A	–	Gr B+C	Gr A (Testing challenge C)	–	Symposium (Challenges B & C)		

Optional evening A+B+C      Optional evening A+B+C      Optional evening A+B+C

\*Important dates: Kick off (24th Apr), Project plan deadline (Wed 3rd May 15:00 pm), Final report deadline (Thu 15th Jun 10:00 pm)

Figure 1.3: EPO-2 timeline in Q4 for groups A, B, and C.

It is also required that 8 hours per week will be dedicated to the project in the form of self-study, this involves preparation for the coming sessions, completing the information on-line course, writing the reports etc. A more detailed schedule is given in Section 1.2.1.

### 1.2.1 Schedule EPO-2 project

Figures 1.4 and 1.5 present the schedule of the EPO-2 project, including the monitoring/control between Student Assistant/Tutor and the group, shown in the project development control column. The main document deliverables of your group will be the project plan as well as the final report. The submission deadline for the project plan is Wednesday, 3rd May 2023 at 15:00 PM. The deadline to submit the final report is Thursday, 15th June 2023 at 10:00 PM. Note that the final report submission folders for the reports will be open on Brightspace.

### 1.2.2 Presence

The presence is compulsory for **all** the scheduled lab sessions of the project. Presence during non-scheduled sessions is based on the arrangements made within the group and with the Tutor, this is discussed on a case by case format and should not be considered as an *extra ordinary* solution. A presence list of all the members composing the group is maintained by one of the group members designated as Secretary, who will communicate it with the Student Assistant / Tutor to maintain the on-line presence list updated. In the event that you are unable to attend one session (i.e., illness, etc.) **please, report it as soon as possible to the group and the tutor.**

When you have missed a session, you will have to arrange extra time to carry out the scheduled project activities. The group is responsible to create rules and schedules related to the missed work. These rules are included in the *House Rules*; the group will compose the house rules during the first EPO-2



lab=lab activities self=self study ma=monitoring activities

Task	Topic	Instruction method	Estimated hours of effort		Project development control
Task 1	Project introduction, presentation of the L.O.(s), structure and goal of the project. Description of the evaluation procedure, the deliverables, the challenges and final presentation. Usage of multimedia.	lab	1	work carried out per chapter where all group is involved	
	Chapter Getting to know the robots, Hardware and Software.	lab	5		
	Chapter Operating and measuring with the robot.	lab	8		
	General discussion feedback on project	ma	1		Group knows the basic elements of the robot, the course objective and requirements.
Task 2	Introduction of week goals feedback from previous week Gantt Chart*	ma	2		
	Finalize routeplanner in C	lab	6		
	Prepare project plan with deliverables/milestones	self/ma	5		Project plan to be presented at beginning of week 3.
	Chapter Intro to VHDL top level def.	lab	3		Group can properly interface with the Spartan board and the Xilinx environment.
Task 3	Present project plan with deliverables/milestones Gantt Chart*	ma	2		Project plan presented
	Chapter The time base	lab	6		
	Chapter The FSM machine input buffer and motor driver	lab	8		Test of basic line follower capabilities
Task 4	Introduction of week goals feedback to project plan Gantt Chart*	ma	1		Feedback to project plan
	Chapter Advanced FSM,	lab	6		
	Chapter Wireless communication, UART	lab	3		Test communication and line follower. Define with tutor and SA key components to improve.

Figure 1.4: EPO-2 project schedule for the week 4.1 to week 4.4.

afternoons and present them to the Tutor for approval, before uploading them to the group's Brightspace folder.



#### Attention

Excessive (unauthorized) absence may result in not being able to successfully complete the EPO-2 project or even exclusion from further participation in the lab for the current academic year.

### 1.2.3 House rules

The house rules should cover, for example, the following points: What do you do with latecomers? How can absentees catch up with their work? What do you do when group members do not work (properly)?

Failure to comply with these rules is first addressed and discussed within the group. If this does not lead to a successful solution the problem is discussed with the tutor and ultimately with the EPO-2 coordinator.

The tasks to be carried out by each member are discussed and agreed during the team meetings. A member selected as *chairman* of the meeting prepares a small agenda of the items to be discussed. The secretary prepares, after the meetings, the minutes as well as the update of the group's **Gantt Chart**. Then, he/she circulates these documents among the group members and if requested to the tutor. Note that the updated Gantt Chart shall be provided to the tutor for the monitoring activities at the beginning of each week in the dedicated Gantt chart assignment folder of the Brightspace. The

*lab=lab activities self=self study ma=monitoring activities*

Task	Topic	Instruction method	Estimated hours of effort	Project deliverables and status
Task 5	<i>Introduction of week goals feedback from previous week Gantt Chart*</i>	ma	2	<i>Team members from different sub groups should present the work of their colleague, random test by SA or Tutor.</i>
	Develop the mine sensor	lab	6	
	Optimize design and employ sensor	lab	4	
Task 6	<i>Introduction of week goals feedback from previous week Gantt Chart*</i>	ma	2	<i>Team members from different sub groups should present the work of their colleague, random test by SA or Tutor.</i>
	Improve your design	lab	6	
	Improve your design and first challenge tests	lab	4	
Task 7	<i>Feedback from previous week, mini demo from team members Gantt Chart*</i>	ma	2	Test sensor. Define with tutor and SA key components to improve.
	Improve your design	lab	3	
	Write final report	self/ma	7	
	Improve your design	lab	4	
Task 8	<i>Feedback from previous week, mini demo from team members Gantt Chart*</i>	ma	2	Define with tutor and SA key components to improve.
	Improve your design	lab	3	
	Final report	self/ma	7	
	Final challenge tests	lab	4	
Task 9	<i>Feedback from previous week, mini demo from team members Gantt Chart*</i>	ma	2	Define with tutor and SA key components to improve.
	Final tune-ups	lab	3	
<b>Final presentations week</b>				Project development control

*work carried out in sub groups in the team*

Figure 1.5: EPO-2 project schedule for the week 4.5 to week 4.9.

chairman and secretary fill their roles in accordance with Chapter 7 of the book R. Grit, *Project Management*, 6<sup>th</sup> edition, Noordhoff, Groningen.

### 1.3 Working method for the EPO-2 project

In the following sections some indications are given on the working method during the EPO-2 project.

### 1.3.1 Project work

The EPO-2 project is a team project, as such it contains various of the dynamics which are common to small and medium scale project work. The main goal that you as a team will have to master, is to properly divide the tasks in order to *speed up* and successfully complete the assignments. Note that the well-defined Gantt Chart, which is being continuously updated during the project, will help the group achieve this goal (see section 1.3.4). This will introduce some *specializations* within the group. At the same time it is crucial for every sub-group to maintain a clear knowledge of the overall project in order to avoid the common mistake of developing sub blocks which do not properly interface, i.e., wrong format or type of variables between the wireless interface and the code to find the route.

Project work is done according to the methods described in the book of **R. Grit, Project Management, 6<sup>th</sup> edition, Noordhoff, Groningen.**

### 1.3.2 Group division

From week 4.1 until the end of the project your EPO-2 group should be organized in different sub-groups, each focused to further develop one aspect of the robot. This is done in order to reach the competition with a robot capable of successfully completing all the challenges.

### 1.3.3 Responsibilities

The team as a whole is responsible for the timely submission of the intermediate and final deliverables through the submission folder in Brightspace. The team takes the more important decisions jointly, while the aspects that pertains to the development of a smaller component/block are usually taken by a smaller group, i.e., the development team of that unit.

The tutor monitors the overall process and provides feedback and comments also on the intermediate report. The tutor only observes from a distance the development of the project which is lead by the team.

As a team member you should, at any time, be aware of the status of your work as well as that of the entire group.

Important for your learning process is asking questions. You can ask questions to yourself, your fellow students, but also to your tutor and student assistant. Do not sit with a question, but try to formulate it and to get an answer: *active learning is constantly asking questions.*

### 1.3.4 Project management - planning

To successfully carry out the EPO-2 project you will need to manage and coordinate the team resources. To achieve that you will have to plan the division of activities among the various team members. You will create a plan (ie, Gantt Chart) at the beginning of week 4.2 (for further information, also see 1.2.3). The plan will be reviewed by your tutor and you will receive feedback. The project plan will be presented and monitored using Gantt Project (**GanttProject Latex**). The plan should contain the division of activities among the members, the indications of when a given product should be ready and it

is important to indicate during the plan key moments to evaluate the progress. These moments are often referred to as *milestones*. Further guidelines for the development of a good plan can be found in the book of Grit, Chapter 5.

### 1.3.5 Place and time

The EPO-2 project is carried out in the *Tellegen Hall* of the low building of the EWI faculty (building 36). Here you can find facilities for meetings, performing the measurements and experiments, use of the Internet (e-mail), word processing, etc.

During the scheduled afternoons the lab rooms are open from 13:45 to 17:30. In the morning sessions, it is open from 8:45 am to 12:30 am. These are, in principle, the working hours unless otherwise agreed upon with the tutor and the student assistant. During the scheduled afternoons the work is performed under the supervision of a tutor and the student assistants.

### 1.3.6 Equipment and facilities

To carry out the EPO-2 project the following items and facilities are available:

- Experimental workplace with:
  - PC with Matlab, Microsoft Office and internet access;
  - soldering iron and hand tools;
  - oscilloscope;
  - multimeters;
  - signal generator;
  - DC power supply;
- Material:
  - boards;
  - resistors, capacitors, inductors, and other necessary components;

These materials are available through the student assistants and/or tutors.

## 1.4 Final mark

The final grade will be calculated from the following components:

1. Final report 30%, this is a group mark;
2. Robot prototype achievements during challenges 30%, this is a group mark (study carefully the symposium rules on EPO-2 Brightspace);
  - Robot can follow the line (6)
  - Robot successfully completes challenge 1 Reaching Station 1 (+0.3)
  - Robot successfully completes challenge 1 Reaching Station 2 (+0.3)
  - Robot successfully completes challenge 1 Reaching Station 3 (+0.3)

Robot successfully completes challenge 2 Reaching Station 1 (+0.6)  
 Robot successfully completes challenge 2 Reaching Station 2 (+0.6)  
 Robot successfully completes challenge 2 Reaching Station 3 (+0.6)  
 Robot successfully completes challenge 3 (+1.3)

3. Final group presentation 25%, this is a group mark;
4. Q&A during final presentation 15%, this is a group mark;
5. + or – final grade corrections based on peer review, this is an individual correction.

### 1.4.1 Final assessment

At the end of the fourth quarter the EPO-2 project will come to the final assessment procedure which determines the final mark (counting between report, presentation and Q&A). The final assessment procedure of EPO-2 consists of two parts:

- The EPO-2 project symposium and a robot competition, during week 4.9 (for further information about the competition and the rules, see Appendix A).
  1. During the robot competition, your Robot will be tested for the various challenges on the on the competition field. Its ability to complete the challenges and the times it requires to do so, compared to the other teams will define its score. The best two teams in the Smart Robot Challenge 2023 will receive a prize.
- In week 4.9 each project team will have to defend their solution of the project task in front of a defense committee. The defense will take about 60 min and will consist of:
  1. oral presentation ( 15-20 min) - each team member 4-5 min;
  2. answering to the questions (15-20 min) - each team member should be able to answer questions concerning any part of the robot design and the group process;
  3. peer review (10 min) - filling in score sheets anonymously to judge contribution/achievement of each team member.
  4. The feedback and grades from the assessors 10 min.

Input for this defense is the final report which need to be submitted on time. Please submit the final version of your report to the assignment folder in EE1L21 Brightspace.

Be sure to consult your report with your tutor/student assistant frequently to get feedback and achieve sufficient quality.

## 1.5 Deadlines of deliverables

It is necessary that the deliverables: the final report and intermediate products required (when it applies) are delivered according to the schedule (deadlines) indicated in Figures 1.4 and 1.5, failing to do so might incur in a final grade penalty.

## 1.6 Project organization

1. Code EPO-2 project: EE1L21
2. Practicum coordinator: Dr. M. Spirito, room HB 18.260, e-mail: m.spirito@tudelft.nl
3. Tutors:
  - Ir. Egbert Bol, e-mail: E.W.Bol@tudelft.nl
  - Dr.ir. A.J. van Genderen , e-mail: a.j.vanGenderen@tudelft.nl
  - Dr. S. Izadkhast, room LB 01.250, e-mail: s.izadkhast@tudelft.nl
4. Study material:
  - R. Grit, *Project management*, 6<sup>th</sup> edition, Noordhoff Uitgevers, Groningen, ISBN 978-90-01-79093-6

## Chapter 2

# Getting to know the Robot

### Learning Objectives

At the end of this session you will have knowledge of:

- the structure of the robot,
- the power source of the robot,
- the type and functionality of the sensors,
- the servo motors of the robot,
- the wireless communication board,
- the software to realize digital control.

### 2.1 Introduction

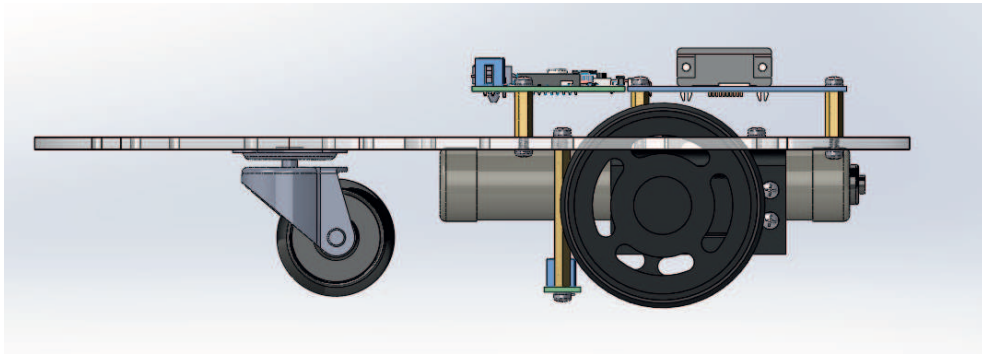
In this chapter you will familiarize with the various elements of the robot. You will first examine all the hardware components of the robot, then the software to control the robot, and finally perform a simple operational test to check the proper functionality of your robot.

### 2.2 Overview of the robot

The robots (see figure 2.1) are composed of a number of different parts, check carefully that you have all the following items available:

- Chassis from a 4 mm thick perspex plate
- GWS continuous rotation servos with 3-pin Futaba style connector
- Swivel caster wheel with 41 mm diameter (See [www.conrad.com](http://www.conrad.com))
- ABS wheels compatible with the servos of 2,56" diameter

- NiMh accu pack 7.2 V, 2200 mAh, with Tamiya connector
- Basys2 FPGA development board
- Robot utility board
- Line tracking sensor board
- ZigBee module XBee S1 from [www.digi.com](http://www.digi.com)
- Prototyping board to implement interface circuitry for the mine-detection sensor (available from week 4.2)



**Figure 2.1:** The robot

A simplified block diagram of the robot is shown in figure 2.2. The sensors provide the inputs to the digital controller, implemented on the FPGA board. The digital controller provides the signals to the motors to operate the robot.

Three light-sensitive sensors are placed in a row on the sensor board. These sensors can measure the reflection from the ground (i.e. light or dark surface). In this way, it is, for example, possible to detect a black line on a white background.

The servo motors are controlled by means of pulse width modulation (PWM). Each motor can be set independently in one of the following operation modes: stop, forward and backward rotation.

The Basys2 FPGA development board contains a Xilinx Spartan-3E XC3S250E FPGA. On the FPGA you will implement a series of hardware modules, which will form all together the digital control of the engines. In order to implement these modules you will use the hardware description language VHDL and the Xilinx ISE design software.

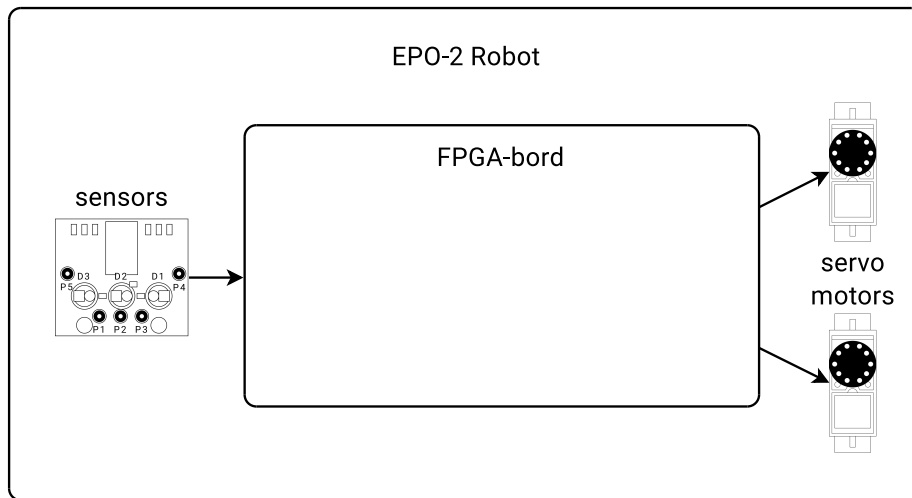
## 2.3 Robot hardware: details and source files

### 2.3.1 Robot Components

### 2.3.2 Chassis

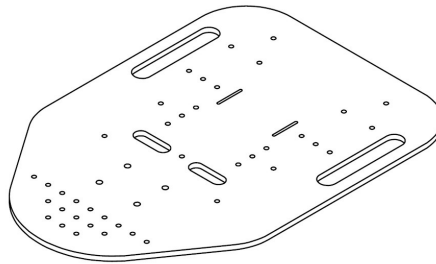
SURFdrive file: Chassis technical drawing





**Figure 2.2:** Simplified block scheme of the robot

The robot chassis is milled out from a 4 mm thick white perspex plate.



**Figure 2.3:** The robot chassis

### 2.3.3 GWS servos

The GWS S35 STD continuous rotation servo with Futaba style connector are used for direct wheel drive. This is a standard size analog servo. The electronics of this servo converts the incoming control pulse into a PWM signal driving a DC motor through an H-bridge ([http://en.wikipedia.org/wiki/H\\_bridge](http://en.wikipedia.org/wiki/H_bridge)). Dimensions: 39.5 mm x 20.0 mm x 35.6 mm Weight 42 g Control pulse: 0.9 - 2.1 ms (positive logic) @ 50 Hz Supply voltage: 4.8- 6.0 V (e.g. direct supply from 4 or 5-cell NiMh/NiCd accu pack).

### 2.3.4 Wheels

ABS plastic wheels with a diameter of 65 mm are suitable for direct attachment to the servos. Rubber treads are fitted for extra traction.



Figure 2.4: GWS S35 STD continuous rotation servo



Figure 2.5: ABS plastic wheels

### 2.3.5 NiMh accu pack 7.2 V, 220 mAH

A 6-cell NiMh battery pack (Nosram art. nr 99260) with Tamiya connector is used for the robot power supply. In this pack, sub C size cells are used. Each project team will get 2 accu packs and is responsible for their proper maintenance. In each project room a dedicated charger (Graupner Ultra Trio Plus 14) and charge cables are available to charge or discharge your accu packs. Please read the user manual before use.

#### Important rules:

- Never short-circuit the battery pack by either, accidentally or intentionally bringing the terminals in contact with another metal object! NiMh cells have (depending on the cell type) very low internal resistance and can provide very high currents.
- Only use dedicated equipment to charge your battery packs!
- Never let the charger unattended when charging with high currents!

- **Never charge an already fully-charged NiMh battery!** The automatic end-detection program will not work which will result in battery over-heating and possibly in explosion. Battery over-heating shortens its life-time.
- Charge current of 1 C (i.e. 2 A for 2000 mAh NiMh battery; 2.8 A for 2800 mAh battery) should be used for charging. In both manual or automatic mode, the charger logic will detect the voltage increase when battery reaches its fully charged state and stop the charging. Some cell types allow higher charge currents (up to 1.6C). Always read the cell data sheet to find the maximum value allowed. Lower charging currents are not desirable as this will make end-detection less reliable.
- Only use dedicated charging cables with low resistance for charging. This helps the charging logic to properly detect the end of charge.
- Discharging with continuous currents of 5 C to 15 C are possible (depending on the cell type).
- Avoid deep discharge (below 1 V/cell) which can cause polarity reversal and shorten the battery useful life time. This means that you have to always physically disconnect the battery pack when not in use and a reliable cut-off mechanism during operation should be applied.
- Store your NiMh battery pack in a discharged state (1 V/cell).
- Long time storage should be done at decreased temperatures ( $-20^{\circ}\text{C}$  to  $+10^{\circ}\text{C}$ ).
- When stored for long time at room temperature, apply maintenance charge/discharge cycle every 4 weeks.

### 2.3.6 Basys2 FPGA development board

Basys2 FPGA development board from Digilent (<http://www.digilentinc.com>) is the central part of the robot. It contains a Spartan3E family XC3S250E FPGA chip from Xilinx in a CP132 package (<http://www.xilinx.com>). For FPGA chip documentation see: <http://xilinx.com/support/documentation/spartan-3e.htm> For Basys2 documentation see: <http://digilentinc.com/Products/Detail.cfm?Prod=BASYS2>

### 2.3.7 Robot utility board

**\*Go to Brightspace > Table of Contents > Robot hardware > Robot utility board (See Utility board schematics)**

The robot utility board provides an interface to the Basys2 board:

- 5 V switch-mode regulator
- 3.3 V regulator for Zigbee module
- battery voltage measurement with LED indicators

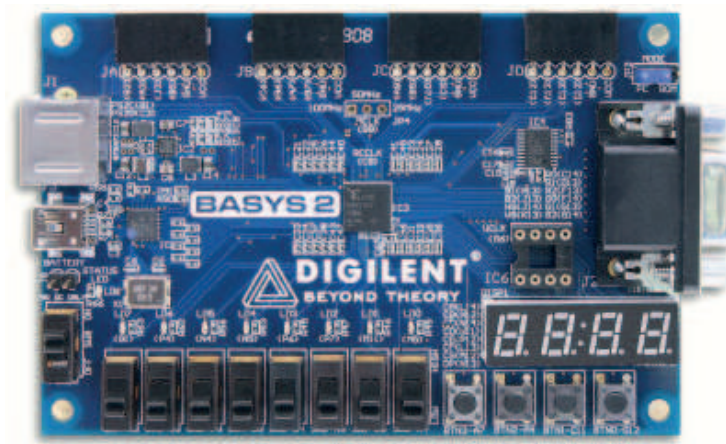


Figure 2.6: Robot utility board

- connectors for the servos, line tracking sensor board, battery, Zigbee module
- LED indicators for the line tracking sensor status

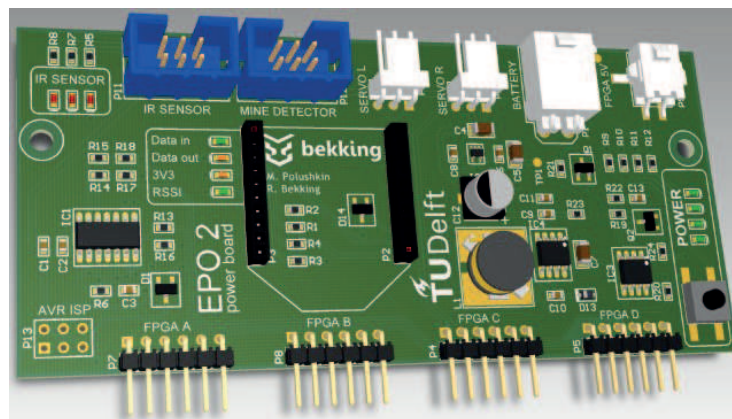


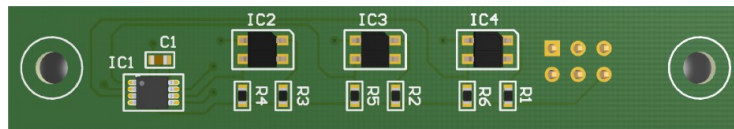
Figure 2.7: Robot utility board

### 2.3.8 Line tracking sensor board

The line tracking sensor board is based on a QRE1113GR reflective object sensor from Fairchild Semiconductor and a Schmitt-trigger inverter.

### 2.3.9 XBee module S1

The Xbee 802.15.4 low-power module with wire antenna from <https://www.digi.com/> is used for wireless communication between the robot and PC/laptop. For its configuration and testing, X-CTU software utility updated with



**Figure 2.8:** The line tracking sensor board

corresponding drivers is required: X-CTU installer and USB drivers - <http://www.digi.com/support/productdetail?pid=3352> The current XBee firmware is ver. 10ec. All X-CTU drivers can be downloaded from digi.com ftp site: <ftp://ftp1.digi.com/support/firmware/update/xbec/> The connection between the XBee module and the laptop is realized using a mini-USB cable. A virtual COM port driver is required to access the XBee module from the X-CTU utility. This can be downloaded from <http://www.ftdichip.com/Drivers/VCP.htm>; Windows compatible installer ver. 2.08.28. Virtual COM port drivers cause the USB device to appear as an additional COM port available to the PC. Application software can access the USB device in the same way as it would access a standard COM port.

### 2.3.10 Prototyping board

The protoboard is used to implement the mine-detection sensor interface circuitry. Moreover, it already contains a piezo loudspeaker to provide acoustic output.



**Figure 2.9:** The protoboard

## 2.4 Robot Software: details and source files

### 2.4.1 FPGA programming file epoztest

This FPGA programming file can be used to test the basic functionality of an EPO2 robot:

- line tracking (SWo can be used to stop the servos)
- sending/receiving of a byte between two robots (or a PC and a robot) using a paired set of XBee modules; btno is used to send the byte value set by SW7 -SWo; the value of the received byte is shown by the discrete leds LD7 - LDo.

### 2.4.2 User constraints file

The epoztest.ucf file can be used in your EPO2 robot projects to define the FPGA pin assignment. Just comment/uncomment the pins you want or don't want to use.

### 2.4.3 Xilinx ISE Design Suite

The latest version of the Xilinx ISE design suite (currently ver. 14.7; Windows or Linux) with integrated simulation tool can be downloaded ( 6 GB) from:

<http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html>



The Xilinx ISE design software, which resides on the PCs in the EPO rooms can also be installed on your own PC. You first need to register at the Xilinx website. To activate the tool you need to acquire the free WebPack license. This can be done automatically during the installation process.

## Chapter 3

# Operating and measuring with the robot

### Learning Objectives

During this session you will:

- understand the operation of the robot components: the battery, optical sensors and engines;
- understand and be able to interpret the accuracy specification of measuring equipment;
- be (re)acquainted with the concept of measurement uncertainty.

And you will be (re)acquainted with the instrument on your table bench: multimeter, oscilloscope and function generator.

### 3.1 Introduction

In this session we will get acquainted with the hardware of the robot: the battery, optical sensors and the servomotors. These parts will come to good use for the project assignment later in the quarter. The sensors and motors will play a major role in the next sessions, in which you are going to teach the robot to follow a line using VHDL.

We are going to perform measurements on the battery, sensors and motors. We will also measure distances with the robot itself. Hereby you will get (re)acquainted with some commonly used instruments and with some key concepts of measurement technology.

In this session, you'll work together with one or two colleagues. Each group will need a number of components. Check carefully that you have the following items available:

- 1 × robot
- 1 × Ruler to measure the test patterns

Most of these components speak for themselves. If you do not have one of these components, or if you do not recognize a component, ask one of the TA's or your Tutor for help.

Furthermore, you also need the following instruments:

- multimeter
- oscilloscope
- function generator

#### Preparation



- Read this chapter carefully; think about what the purpose of the measurement is; what do you need and what should you do to achieve the expected results.

## 3.2 The battery

### 3.2.1 Theory

The robot is powered by a battery pack consisting of four NiMH cells. This battery pack has a capacity of about 2000 mAh and provides a voltage of about 6 V when it is fully charged. On the FPGA board, the voltage is converted to a stabilized supply voltage of 3.3 V for the FPGA. This is done with a voltage regulator of the type LM1086CS-AD. [9].



### 3.3 The optical sensors

#### 3.3.1 Theory

The line-tracking sensor is composed by three light-sensitive sensors that are placed in a row. Each sensor circuit has a digital output that indicates the degree of reflection of the substrate. When over a black surface (bad reflection), the sensor circuit will set a '0' on the output, over a white surface (good reflection) a '1'. Each sensor circuit has a LED indicator that lights up when the sensor sets a '1' on the output.

Each sensor circuit consists of three parts:

- the sensor and corresponding components
- the signal processing
- the output with LED indication

The light-sensitive sensor itself consists of an infra-red (IR) and a LED (light-sensitive) phototransistor. In normal use the IR LED always lights up. The light that radiates from the IR LED is reflected by the surface (white surface reflects more light than a black surface). The more light is reflected back, the better the conductivity of the phototransistor.

The signal processing is provided by a so-called inverting Schmitt trigger<sup>1</sup>. This ensures that the analog input signal is converted to a digital output signal, and that the output signal is buffered. This is necessary, otherwise the indicator LEDs would influence the output signal of the sensor. A schematic of the sensor is shown in Figure 3.1.

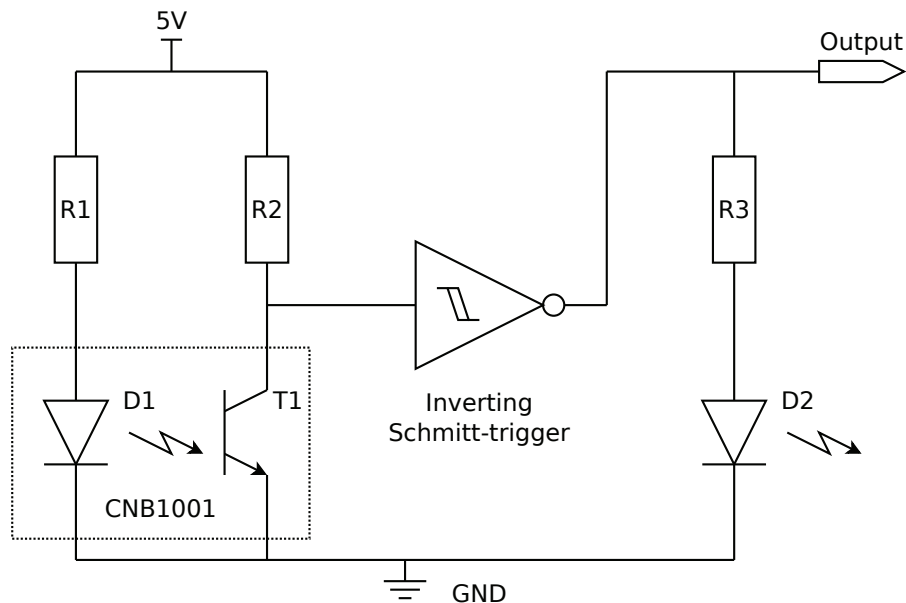


Figure 3.1: Circuit diagram of the sensor

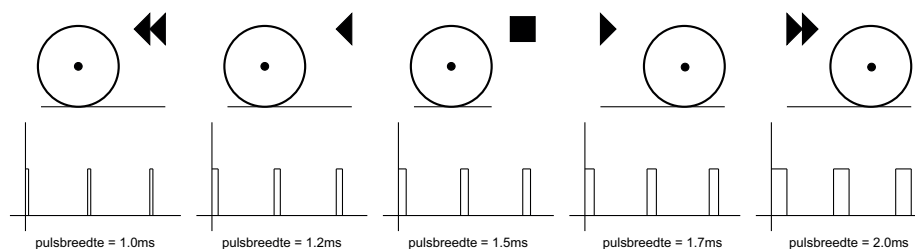
<sup>1</sup>[http://en.wikipedia.org/wiki/Schmitt\\_trigger](http://en.wikipedia.org/wiki/Schmitt_trigger)

If there is no light reflected on the phototransistor, the resistance of the phototransistor is much larger than  $R_2$ . Then a voltage of 5V is at the input of the Schmitt-trigger. As more light is reflected, the resistance of the phototransistor, and thus the voltage at the input of the Schmitt-trigger, decreases. At a given moment the resistance of the phototransistor is much smaller than  $R_2$ . There is then 0 V at the input of the Schmitt-trigger. This voltage is inverted by the Schmitt-trigger. At a white surface (high reflection) the output of the sensor circuit is 5 V and the light of the LED-indication is on.

## 3.4 The servomotors

### 3.4.1 Theory

To move forward, the robot features servomotors, where the servo-part is removed <sup>2</sup>. These motors are controlled by means of a pulse width modulation (PWM). PWM is a technique in which the desired information is encoded in pulses of variable width. The operation in this case is fairly simple: the PWM generator sends a pulse every 20 milliseconds. If this pulse is narrower than 1.5 millisecond, than the motor rotates counter-clockwise, if the pulse is wider than 1.5 millisecond, the motor rotates clockwise. The rotation speed can also be influenced: if the pulse width is around 1.5 milliseconds, the motor will run very slowly or even stop. If the pulse width is significantly different from 1.5 millisecond, the motor will run faster.



## 3.5 Conclusion

In this session you have been introduced to different parts of the robot. By measuring and testing the battery, sensors and motor, you now know how these components behave. You also tested the line follower which you had worked on during the EE1D21 course labs.

<sup>2</sup><http://en.wikipedia.org/wiki/Servomechanism>

## **Part II**

# **Project Smart Robot Challenge**

## Chapter 4

# Maze router

Required preparation:

- Read this chapter and try to understand the Lee algorithm

### Learning Objectives

During this assignment you will learn how to solve a technical problem through the development of a C program. You will practice how an appropriate algorithm can be implemented, including a suitable datastructure.

### 4.1 Introduction

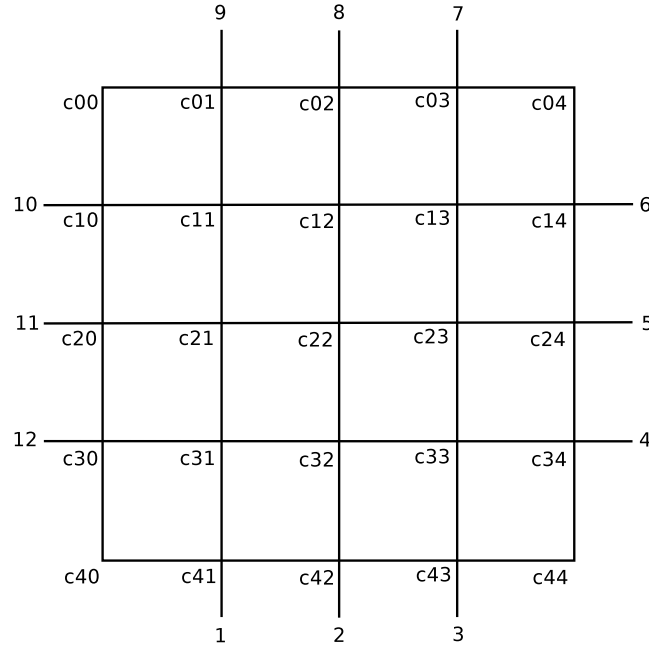
During this assignment you will develop a maze router that can be used for the robot of the EPO-2 project "Smart Robot Challenge". For this assignment, the maze router is created as a separate program with a simple user interface. Later, you will incorporate the program into a larger program that will control the robot.

We will describe an algorithm that can be used to compute the route through the maze, as well as a datastructure to implement the algorithm. The assignment can be done in a group of 2 students.

### 4.2 Assignment

The maze field of the EPO-2 project "Smart Robot Challenge" is given in Figure 4.1. There are 12 stations, numbered from 1 till 12, that act as starting or end point of a route through the maze. There are horizontal and vertical roads between the stations, and the crossings of these roads are labeled with names like  $c_{00}$ ,  $c_{01}$ ,  $c_{02}$ , etc. A road, or edge, between two crossings  $c_{i,j}$  and  $c_{i,j+1}$  is denoted by " $i\ j\ e$ ", where  $i$  and  $j$  are indices, and the letter 'e' indicates "east". Between two crossings  $c_{i,j}$  and  $c_{i+1,j}$  the edge is denoted by " $i\ j\ s$ ", where 's' stands for "south".

The program should take as input a list of blocked edges, followed by a pair of stations between which a route should be computed. The list of



**Figure 4.1:** The maze field. Stations are numbered from 1 till 12, crossings have names c00, c01, c02, etc.

blocked edges is preceded by the number of blocked edges. An example input is given below.

```
2 2 0 s 2 3 e
10 12
```

This example specifies that there are 2 blocked edges, between respectively c20 and c30, and between c23 and c24, and that a route has to be computed from station 10 to station 12. The output of the program should be a free route between the 2 stations, given as a list of the crossings that are passed on the route. For the example input, the output should be one of the following lines:

```
c10 c20 c21 c31 c30
c10 c11 c21 c31 c30
```

### 4.3 Algorithm

There are several algorithms known for shortest path computation, see e.g. [13]. In our case we are dealing with a rather simple map, which is rectangular and does not have a lot of nodes (points) and edges (roads). Therefore we propose to use also a rather simple algorithm that is straightforward to implement. This is the Lee or wavefront algorithm, see [14] and [15], that was originally developed to solve routing problems for the design of integrated circuits. Note that it is not compulsory to use this algorithm to solve this assignment, and that you are free to choose another algorithm, as long as it provides a valid solution for all possible input combinations.

To describe the Lee algorithm, we model the “Smart Robot Challenge” map from Figure 4.1 by an array of  $13 \times 13$  cells as shown in Figure 4.2. Each station, crossing and edge in the original map is thereby represented by a cell, such that when items are adjacent in the original map, their corresponding cells in the array are also adjacent. The remaining cells in the array model the empty space in the original map. Cells corresponding to stations, crossings or unblocked edges are initialized with a value 0. Cells corresponding to empty space or blocked edges are initialized with a value -1.

					9	8	7					
	-1	-1	-1	-1	0	-1	0	-1	0	-1	-1	-1
	-1	-1	-1	-1	0	-1	0	-1	0	-1	-1	-1
	-1	-1	0	0	0	0	0	0	0	0	0	-1
	-1	-1	0	-1	0	-1	0	-1	0	-1	0	-1
10	0	0	0	0	0	0	0	0	0	0	0	0
	-1	-1	0	-1	0	-1	0	-1	0	-1	0	-1
11	0	0	0	0	0	0	0	0	0	-1	0	0
	-1	-1	-1	-1	0	-1	0	-1	0	-1	0	-1
12	0	0	0	0	0	0	0	0	0	0	0	0
	-1	-1	0	-1	0	-1	0	-1	0	-1	0	-1
	-1	-1	0	0	0	0	0	0	0	0	0	-1
	-1	-1	-1	-1	0	-1	0	-1	0	-1	-1	-1
	-1	-1	-1	-1	0	-1	0	-1	0	-1	-1	-1
					1	2	3					

**Figure 4.2:** Array representation for the maze field, suitable for applying the Lee algorithm

Next, the Lee algorithm is applied for the array as follows. In the algorithm, a neighbor cell is defined as a cell that is directly to the right, to the left, above or below the cell that is considered (so the cells that diagonally touch each other are not considered neighbor cells).

```

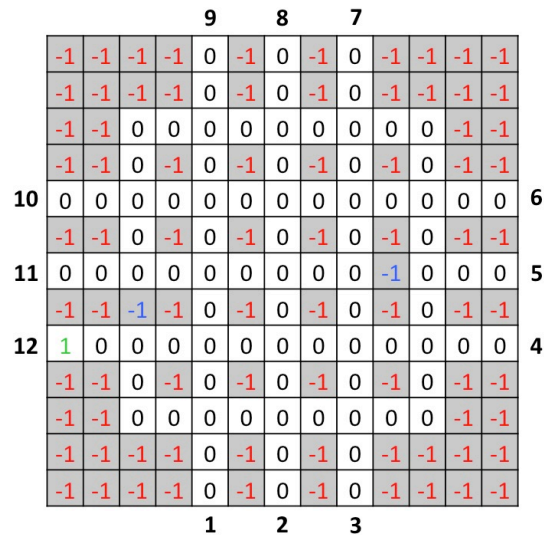
i = 1;                                     /* Expand */
Mark target cell with value i
while (start cell not yet marked with value > 0) {
    Mark all neighbor cells of cells marked with i, with i+1,
    if they have a value 0.
    i = i+1
}

Go to the start cell                       /* Trace back */
while (target cell not yet reached) {
    Go to a neighbor cell that has a lower value.
}

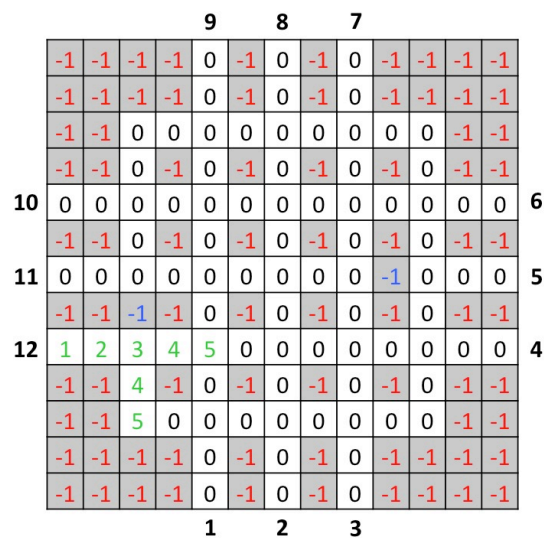
```

The contents of the array after expansion steps 1, 5 and 13 of the algorithm, are shown in respectively Figure 4.3, Figure 4.4 and Figure 4.5. The path that

is followed during the trace back step of the algorithm is also shown in Figure 4.5. Note that an alternative path is via c01 instead of via c13.



**Figure 4.3:** Expansion step 1 of the Lee algorithm



**Figure 4.4:** Expansion step 5 of the Lee algorithm

## 4.4 Datastructure

The datastructure that is used to implement the above algorithm can be quite simple. We need to maintain a 2-dimensional array of integers of size  $13 \times 13$ .

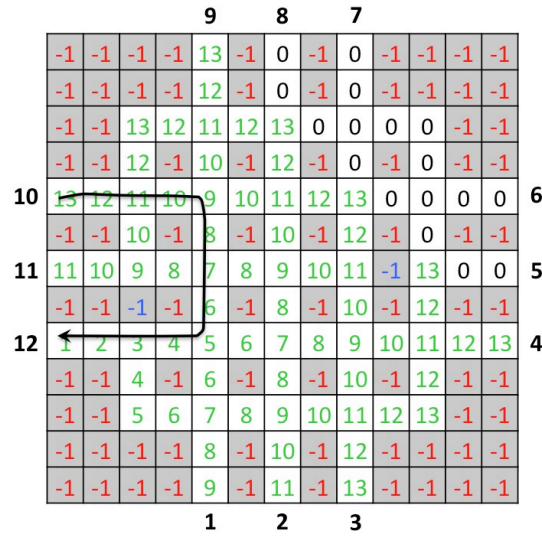


Figure 4.5: Expansion step 13 of the Lee algorithm, as well as a path during trace back.

```
int maze[13][13];
```

The program then typically will start by initializing the array according to the topology of the maze. After that the blocked edges should be read and represented in the array, followed by reading the start and target station, and finally the application of the Lee algorithm.

## 4.5 Development

As a development strategy it is suggested to first implement the Lee algorithm itself, and not yet read the specified user input. Test the code by directly initializing the datastructure (matrix) in the C code. To debug the code, it may be useful to write a function that prints the contents of the datastructure (matrix). When everything is working correctly, extend the program so that it meets the input/output requirements as mentioned above.



## Chapter 5

# Wireless Communication between PC and Robot

### 5.1 Introduction

The project requires that an application written in C, that runs on a PC, guides the robot through the maze. For this we are going to use XBee modules to create a wireless communication link between the PC and the robot, as shown in Fig. 5.1. XBee modules communicate with each other wirelessly through the Zigbee protocol, and communicate with external hardware using the UART protocol. The Zigbee protocol is outside the scope of this course.

Sec. 5.2 will give a comprehensive introduction to the UART protocol to brush up your knowledge. Sec. 5.3 explains how to let the robot communicate with an XBee module. Sec. 5.5 explains how to let your PC communicate with an XBee module. And Sec. 5.4 explains how to configure the XBee modules.



**Figure 5.1:** Data transfer path between robot and PC.

After this chapter you should be able to:

1. understand how this serial communication link works,
2. understand how the UART protocol works, and how it is used in the communication link,
3. configure and use an XBee module,

4. write a basic C-program that sends/receive bytes over the XBee wireless communication link.

## 5.2 The UART Protocol

In this section first the basics of asynchronous serial communication are introduced. Then the Universal Asynchronous Receiver/Transmitter (UART) circuit as an interface between the system CPU and serial communication line is described and implemented using VHDL. Working of the UART circuit is demonstrated by sending and receiving of 1 byte between two SPARTAN-3 development boards.

### 5.2.1 UART: What and why?

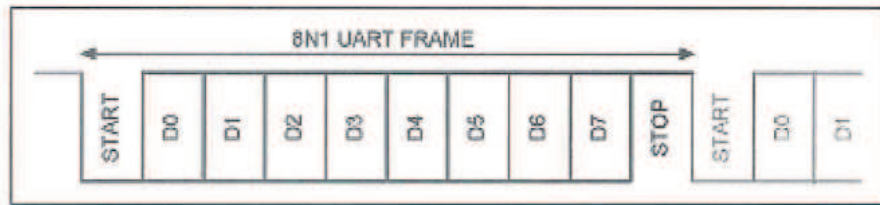
In telecommunication and computer science, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel. This is in contrast to parallel channels. Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties make parallel communication impractical. In many cases, serial is a better option because it is cheaper to implement. Many ICs have serial interfaces, as opposed to parallel ones, so that they have fewer pins and are therefore less expensive.

A Central Processing Unit (CPU) of a computer performs arithmetic operations on binary numbers which are represented by groups of bits (8 bits = 1 byte, 16 bits = 2 bytes, etc.). In the beginning of the PC era dedicated interface circuits called Universal Asynchronous Receiver/Transmitter (UARTs) were developed that translate data between parallel and serial forms. Nowadays this piece of hardware is commonly integrated on one chip together with the CPU.

The term "universal" means that the protocol works, regardless of data-rate (baud-rate) and used signaling methods (signal levels, single-ended vs. differential signaling, etc.). UARTs are commonly used in conjunction with communication standards such as EIA RS-232, RS-422 or RS-485. Because of their universality the data format and transmission speeds are configurable, and the actual electric signaling levels and methods (such as differential signaling, etc.) typically are handled by a special driver circuit external to the UART.

### 5.2.2 Asynchronous serial communication basics

The term "asynchronous" means that the clocks of the transmitting and receiving systems are not synchronized. The receiving system does not know when exactly a certain bit of information is arriving and needs to be sampled. To alleviate this problem a simple communication scheme was developed where the useful information, i.e. serial stream of bits is regularly sandwiched between predefined signals levels (e.g., 1 START and 1 STOP bit) as shown in Figure 5.2. These additional signals does not carry useful information but allows for synchronization of the receiver.



**Figure 5.2:** A typical asynchronous serial communication data frame

UART modules typically go through the following steps when exchanging data:

1. When there is no communication, the communication line is in the BREAK state with signal level '0'.
2. Before the communication starts, the communication line goes into the IDLE state with signal level '1'. This indicates to the receiver that communication can start any moment.
3. The actual communication starts with a START bit having signal level '0'. The UART receiver synchronizes itself to the falling edge of this START bit. If the transmitting and receiving systems agree beforehand on how long each data bit lasts (defined by signal baud rate), then the receiver is able to sample with sufficient accuracy certain number of incoming DATA bits that immediately follow the START bit by measuring the time elapsed from the falling edge of the START bit. How many data bits can reliably be sampled in one UART frame depends on the accuracy and resolution of the transmitter and receiver clocks. A data bit will reliably be sampled if sampling is done close to its middle. As the transmitter and receiver clocks are not synchronized the sampling position of each subsequent data bit will more and more deviate from the optimum middle position and eventually the reading of incoming data bits will become incorrect. In practice the number of data bits in one frame is set to 7 or 8. This is sufficient to transfer one ASCII character in one data frame and at the same time poses not so stringent requirements on the accuracy of the transmitter and the receiver clocks.
4. The UART frame is completed with a STOP bit. Its length is set to 1, 1.5 or 2 times of the length of 1 DATA bit.
5. The STOP bit is followed by IDLE signal of any length (zero length is also allowed). Then, the entire process of frame generation can start again. Sampling of the STOP bit can be used for basic check of signal integrity.

Signal integrity can be checked also by inserting a parity bit between the last DATA bit and STOP bit. More common and efficient is however to perform integrity check on higher level using sum check. Here hundred of data frames are sent in one packet to which a sum check is added. The receiver then verifies whether the sum of the values of the given data group equals the sum-check value sent by the transmitter.

In modern systems with crystal derived clocks (typical accuracy of 100 ppm) the integrity of signal transmission is influenced more by the quality of communication link (signal-to-noise ratio, signal dispersion, etc.) than by time asynchronicity between the transmitter and receiver. A UART frame '8N1' (8 DATA bits, no parity, 1 STOP bit) is most commonly used.

Although any length of a DATA bit can be agreed on between the transmitter and the receiver, in practice only values defined by different standards are used. Common baud rates are: 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400, 460800, 921600. Baud is synonymous to symbols per second. In case of old communication techniques where 1 communication symbol can have only 2 stats (i.e., 1 bit), the baud value equals to gross bit rate, e.g. 9600 Bd = 9600 bits/s.

### 5.3 The Robot-XBee link (UART)

An XBee module can be attached by mounting it on the power board of the robot. The board links the pins of the module directly to pins of the FPGA on the robot. To communicate with the XBee module, VHDL code for a UART transceiver. *You don't have to write this yourself. The complete VHDL description can be downloaded from Brightspace!* The code is based on the excellent book *FPGA Prototyping by VHDL Examples* by Pong Chu. The top-level description of the UART transceiver is as follows:

```

1 entity uart is
2 port (
3     clk          : in  std_logic;
4     reset        : in  std_logic;
5
6     rx           : in  std_logic; -- input bit stream
7     tx           : out std_logic; -- output bit stream
8
9     data_in      : in  std_logic_vector(7 downto 0); -- byte to be sent
10    buffer_empty : out std_logic; -- flag '1' if tx buffer empty
11    write        : in  std_logic; -- flag '1' to write to tx buffer
12
13    data_out      : out std_logic_vector(7 downto 0); -- received byte
14    data_ready    : out std_logic; -- flag '1' if new data in rx buffer
15    read         : in  std_logic -- flag '1' to read from rx buffer
16 );
17 end entity uart;
```

The ports on the module's interface can be separated into two parts:

- rx and tx form the UART interface. These ports are connected to the pins of the FPGA that are connected to the XBee module.
- All other ports (except of course clk and reset) make up the "user interface", or the interface with which your VHDL description will communicate with the module.

You will need to understand how the user interface works, but don't need to know how the entire VHDL description of the UART works. Though, a detailed description is provided at the end of this section for interested students.

The user interface can be split up into two parts too: a transmitter, and a receiver part. The transmitter interface consists of the following signals:

- `data_in` is a bit vector containing the byte of data that you want to transmit.
- `buffer_empty` is a signal that is '0' if the UART module is ready to accept new data for transmission, and '1' if the module is busy transmitting data.
- `write` is a signal that enables the UART module to copy the data from `data_in` if '1', and lets the module ignore `data_in` if '0'. It is recommended to make `write` '1' for only one clock cycle to transmit a byte.

The receiver part of the interface consists of the following signals:

- `data_out` is a bit vector containing bytes of data that were received.
- `data_ready` is a signal that goes to '1' if newly received data was available at `data_out`.
- `read`, if set to '1', sets `data_ready` to '0', meaning that you tell the UART module that you've read the data, and that your system can accept new data. Nothing will happen when `read` is set to '0'. It is recommended to make `read` '1' for only one clock cycle to reset the UART receiver.

These interfaces are very similar to ready-valid handshakes, a protocol for data exchange between modules that is very popular in industry. Feel free to look this up!

### Explanation of how the UART VHDL description works

A block diagram of the UART transceiver is shown in Figure 5.3 (note: this image is old and port names may vary, but the working principles are still the same). The module consists of 2 separate stages: one for transmitting and one for receiving bytes. The `rx` and `tx` ports on the left are the serial ports. On the right are the corresponding parallel data ports `data_out` and `data_in`.

Between the transmitter/receiver and the parallel data ports are registers/buffers. You can talk to these buffers by writing to them (tx only) or reading from them (rx only). To do so, set the corresponding `write` or `read` flag to 1 for a single clock cycle. These registers also have a additional flags `buffer_empty` and `data_ready` that basically tell the surrounding modules that the tx stage and/or the rx stage is in the process of (de)serializing data, and therefor temporarily does not allow writing to/reading from its buffers.

**UART receiver** The UART receiver is an FSM consisting of 4 states: idle, start, data and stop. When the FSM is idle and detects the START bit, it enters the start state. From there it counts 8 oversampled UART clock cycles to the middle of the start bit (see Figure 5.4). The 16x oversampled UART clock cycles are counted by the baud generator module in Figure 5.3. This module has a function similar to that of the timebase of the robot. After the start bit the FSM enters the data state, where it counts to 16 for a total of 8 times,

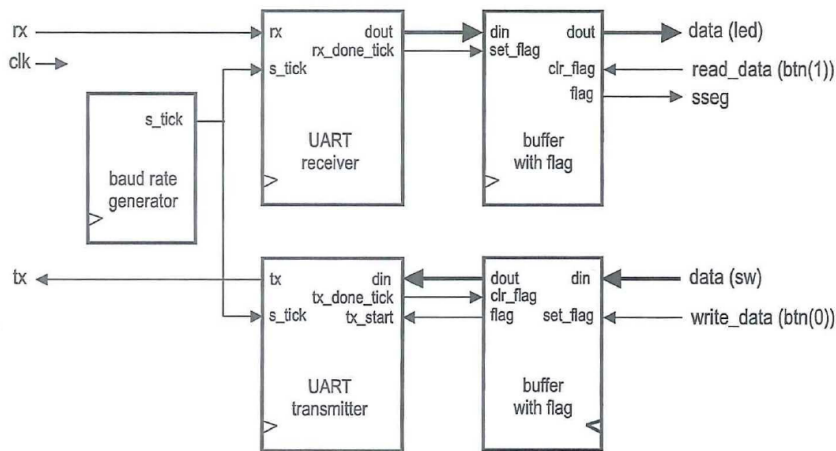


Figure 5.3: Block diagram of a complete UART system

sampling the value of the `rx` signal at the 16th count before. After that, the FSM enters the stop state, counts to 16 one last time, sampling the stop bit, and goes idle again.

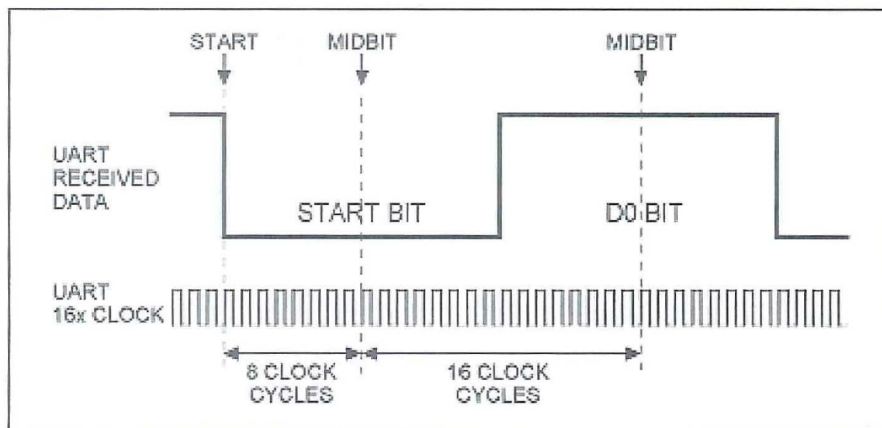


Figure 5.4: A UART frame synchronisation and data bit sampling points.

**UART transmitter** The UART transmitter is essentially a shift register that shifts out data bits at a specific rate controlled by the sampling ticks from the baud rate generator and by an internal counter counting from 0 to 15.

## 5.4 The XBee-XBee link (Wireless)

This chapter is a step-by-step guide to help you get started using XBee wireless modules. You should already be familiar with computer serial ports and UART. We will learn how to configure XBee modules (set the baud rates, PAN ID, serial settings, etc) on your computer using X-CTU.

ZigBee is a low-cost, low-power, wireless mesh networking proprietary standard like Wifi and Bluetooth. XBee modules are bidirectional 2.4 GHz digital radios which have a typical range of about 30-60 meter in a normal home or office. The modules have a 3.3 Volt serial interface with selectable baud rates (1200-230400 bps). After applying power, one can start sending/receiving data over the TX/RX lines. However, when you live in a wireless world with WiFi, bluetooth, cordless phones, game controllers, and tons of other 2.4 GHz devices, it is possible, for packets to get lost.

### 5.4.1 Setup

For this tutorial you need:

- A PC running the Windows 10 OS with the X-CTU software suite installed
- Two XBee Modules with XBee USB Explorer boards;
- Two USB-A → USB-mini B cables;

### 5.4.2 Configuring the modules

Before you can use the XBee modules, you need to configure them with the desired baud rate, PAN ID and Device ID. The best way to do this is through your PC. Hook up the USB-B mini port on the XBee USB Explorer Board module and connect it to your computer. The XBee Explorer Board is a small, red PCB provided in the Tellegen Hall. The board has an FTDI FT232RL chip that converts basic TX/RX signals into USB.

There are two important settings to change. First, you need to come up with a way to address individual modules. When you have a multi-point network, link, or even a different XBee system nearby, you have to make sure you only talk to the right module, and not the others. Another problem is if you have a different XBee (not on your network) somewhere close by that will interfere with yours. DIGI has three ways to separate different networks or links so they can communicate without interference and select which radio in your network you want to talk to.

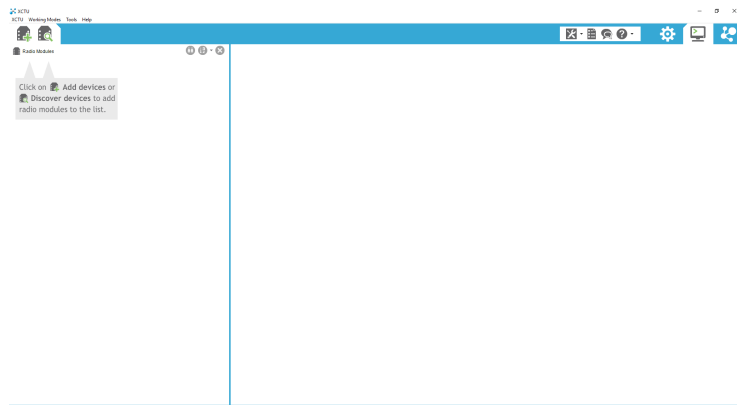
**Using the channel setting** The XBee modules can operate on 16 different frequency bands numbered 11 (0x0B) to 26 (0x1A). So, if you have to separate XBee links in vicinity, put them onto different frequency bands and there will be no interference or communication between the two links. Normally, this setting is used to prevent interference with other 2.4 Ghz devices like microwaves or WiFi. Leave this parameter as is.

**Use the PAN ID** If you have nearby XBee links on the same channel, you can still isolate them by using the PAN ID. This is a 16-bit number that is programmed onto every XBee in your network. When you send a packet to another XBee, it will only process it if the sender's PAN ID matches its own. Essentially, the modules only send/transmit to modules with the same ID.

**MY ID/DL** The above two methods separate entire networks, but if you have a number of XBee's in a multi-point situation (the same PAN network), you still need a way to select which of the other modules to talk to. This is why every XBee can be programmed with a unique 16-bit source address in register MY. By setting the destination register (DL) with the MY value of another module vice versa two modules can communicate.

Once you are connected to your computer, there are two ways to program the module. The first is by using a program called X-CTU, which is released by DIGI (the makers of XBee) for Windows only, not for Mac OSX or Linux. We will use X-CTU in this section. The second is by writing your own interface program, which we will do in section 5.5.

In order to configure a module, plug it into the PC with the USB cable. Start the X-CTU software. This should provide you with the screen of Figure 5.5. There are no devices visible yet. Click on Discover devices to search for connected devices.



**Figure 5.5:** Screenshot of the start screen of X-CTU

After clicking Discover devices, you should see a dialog similar to Figure 5.6. The exact port numbers may differ on your system. On the lab PCs COM1 is an actual serial port. The COM5 USB serial port shown here is represents the XBee module. Tick the corresponding box and press Next.

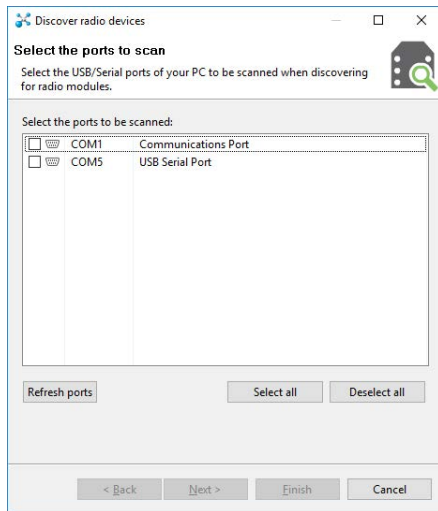
In the following dialog (see Figure 5.7), you can specify serial/USB port parameter in order to narrow or broaden the search. The default settings should be OK:

Baud Rate	9600
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

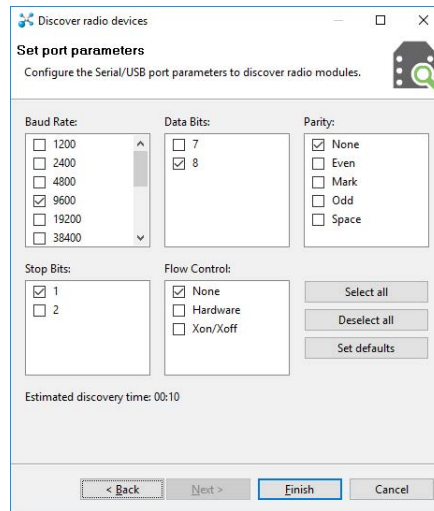
Press Finish to start the search process.

The search will now start, during progression you should see a dialog similar to the screenshot in Figure 5.8.

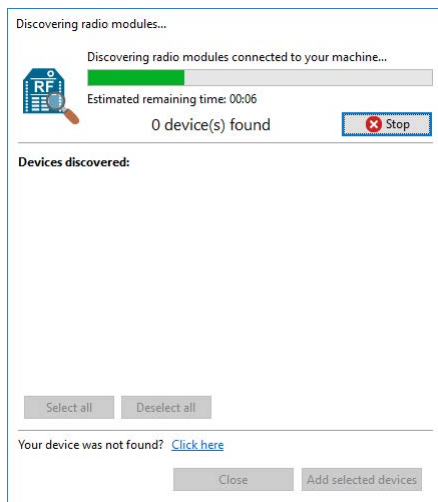




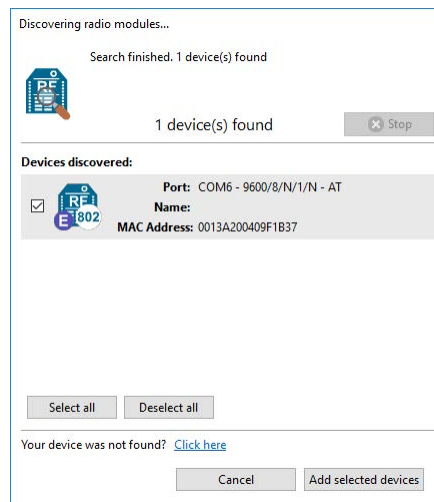
**Figure 5.6:** Screenshot of the Discover radio devices dialog



**Figure 5.7:** Screenshot of the Set port parameters dialog



**Figure 5.8:** Screenshot of the Discover radio modules dialog



**Figure 5.9:** Screenshot of the devices found dialog

The total search time depends on the number of options selected. When the search process has finished, you should see a dialog similar to the screenshot in Figure 5.9. Select the device and click Add selected devices. Now the device should be added to the main window.

### 5.4.3 Configuring the Devices

Now that you've added the device, you can see and modify the current settings by clicking the device, see Figure 5.10. Since we want to setup communications between two devices, connect the

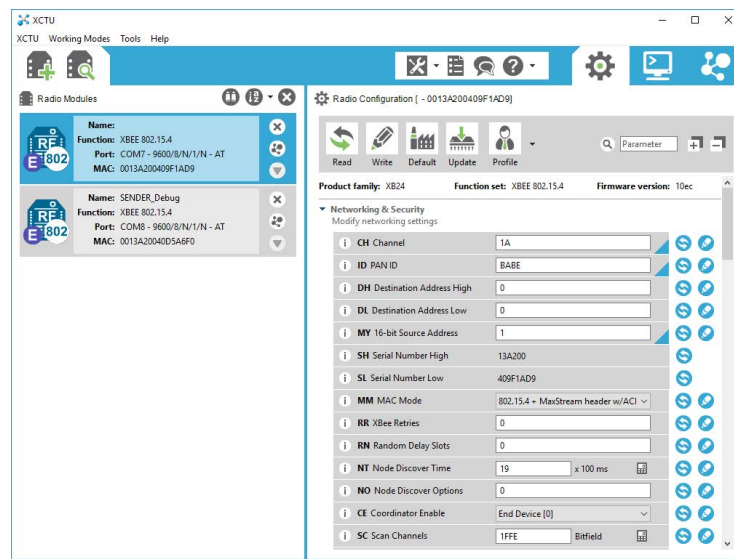


Figure 5.10: Screenshot of XCTU with the device config page opened

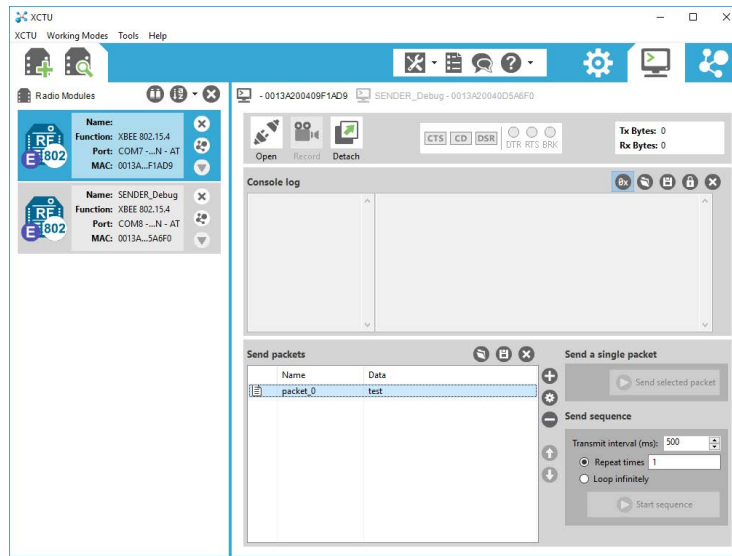
other XBee module as well and repeat the previous steps to add the device in XCTU. Now you can configure both devices to make sure they can communicate:

- Both devices should use the same **CH** Channel setting;
- Both devices should use the same **ID** PAN ID setting;
- Specify the **MY** 16-bit source address of device A and copy the value to the **DL** Destination Address Low setting of device B.
- Specify the **MY** 16-bit source address of device B and copy the value to the **DL** Destination Address Low setting of device A.

Settings that have been modified, are shown with a green triangle in the lower right corner. Save the configuration by pressing the Write icon, the green triangle should now turn blue. Both devices are now configured to communicate.

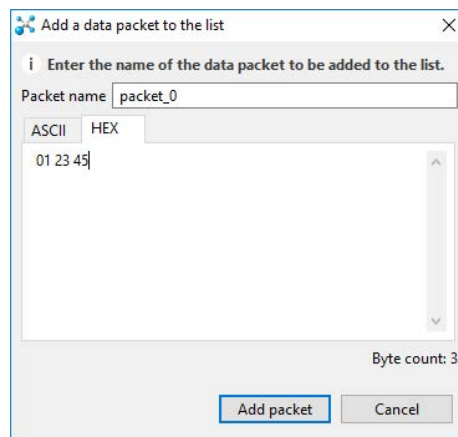
#### 5.4.4 Testing the Communication

In order to test the communication we will use the serial terminal functionality of XCTU. Open the terminal window by pressing the Consoles Working Mode icon in the top left corner of the program (or press **Alt** **C**). This should result in a window similar to the screenshot of Figure 5.11. In the top left corner of the terminal window you see an Open icon. Click the icon to open a connection to the XBee module. Repeat this step for the other device. You should now see two small green terminal icons in the top of the terminal window.



**Figure 5.11:** Screenshot of the terminal windows of XCTU, connected to both devices

Communication in the terminal windows takes place via *packets*. Before you can send any information, you have to create a packet first. Select one of the devices and click the + icon left of the Send selected packet button. This will open the dialog in Figure 5.12.



**Figure 5.12:** Screenshot of the packet definition dialog

You can specify either a text message (tab ASCII) or a string of hexadecimal numbers (tab HEX). Here a hexadecimal string is selected. Create a message of your liking and press the Add packet button to create the new packet definition. Create a packet definition for the other device as well, make sure to use a different message text.

Your newly created packet definitions will appear in a list left of the Send selected packet button. Select a packet and press the Send selected packet button. You should see the send bytes appear in blue on the Console log. If you open the terminal window of the receiving device, the same bytes should appear in red, and vice-versa. This is shown in the screenshot in Figure 5.13.

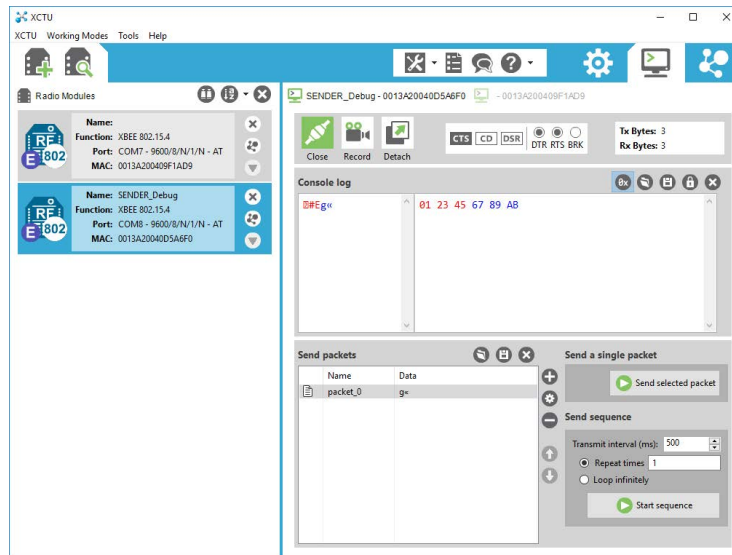


Figure 5.13: Screenshot of the terminal window after sending and receiving packets

If the communication does not work, please verify if all settings are correct. If you do have a proper communication, you can try to test communication with the robot next.

#### 5.4.5 Exercises (optional but recommended)

1. Setup a wireless communication link between 2 PC's with 2 XBee modules and show the communication by sending characters from one PC to the other vice versa using X-CTU.
2. Setup a wireless communication link between a PC and a robot with 2 XBee modules and show the communication by sending a character from the PC using X-CTU to the robot and display the ASCII bit-pattern of the character on the leds of the Spartan3 board.

### 5.5 The XBee-PC link (Windows serial)

Your C program does not have to use X-CTU to communicate with an Xbee module. It can do it through serial programming. Before writing your own serial interface in C read the *Windows Serial Programming* tutorial [7].

**Erratum**

The tutorial has an erratum in section 2 at:

```
DCB dcbSerialParams = 0;
dcbSerialParams.DCBlength=sizeof(dcbSerialParams);

dcbSerial.DCBlength should be dcbSerialParams.DCBlength.
```

Next study the template program given in Appendix B.2. The template code also available on Brightspace. This template code consists of 4 functions:

**initSio()**     Initializes the basic file handle to a COM port. (here COM5)  
**writeByte()**   Writes a single byte to the COM port.  
**readByte()**     Reads a single byte from the COM port.  
**main()**         The main function that opens a file handle to the COM port.

At line 6 of the template code you can configure the COM port name and baud-rate:

```
#define COMPORT "COM5"
#define BAUDRATE 9600
```



You have to change "COM5" string in case your XBee module is connected to another COM port. (check with X-CTU).

### 5.5.1 Setup

For this tutorial you need:

1. Two PC's running Windows 10
2. The Codeblocks C programming environment
3. Two XBee modules
4. Two USB Type A - mini-B cables
5. A robot or Spartan3 board

### 5.5.2 Exercises (optional but recommended)

1. Setup a wireless communication link between 2 PC's with 2 XBee modules and show the communication by sending characters from one PC to the other vice versa using your own written C program .
2. Setup a wireless communication link between a PC and a robot with 2 XBee modules and show the communication by sending a character from the PC using your C program to the robot and display the ASCII bit-pattern of the character on the leds of the Spartan3 board.

## Chapter 6

# Mine Detector Design

During challenges B and C there will be mines (metal discs) on the midpoint of certain lines in the maze. To avoid them, your robot will need a mine sensor that can detect them. This chapter contains design requirements and rough design guidelines of this sensor.

Students are recommended to use the free circuit simulator LTspice during the design. LTspice can be downloaded from [this web page](#).

### 6.1 Working Principle

Sensors are used to transfer information embedded in some physical quantity to the electrical domain. The first questions to ask when designing a sensor are thus: What information am I interested in? And what physical quantity is this information embedded in? In the case of the mine sensor we can say:

- Information: whether there is a metal disk in front of the robot or not
- Physical quantity: proximity (distance) to a metal disk

The next question is: How am I going to convert this quantity into the electrical domain? During the course Amplifiers and Instrumentation (EE1C31) you learned about capacitive and inductive sensors. Techniques were discussed which can be used to change the value of a capacitor or an inductor in the presence of a metal object.

The working principle of the mine sensor is to use one of these techniques to change the properties of an LC-resonator circuit depending on whether a mine is close by or not. This changes the resonance frequency of the resonator. The frequency difference can be detected using a digital circuit on the FPGA.

### 6.2 Design constraints

- The robot is driven by a 5 Volt DC battery, which is the only external bias value that can be used for the design project.

- The only type of op-amp you are allowed to use, is National Semiconductor's LM358. The SPICE model (.mod file extension) which can be used in LTspice for simulation, can be found on Brightspace.

### 6.3 Design Tips

- Decide whether capacitive or inductive sensing is more effective using knowledge and lab skills from the Amplifiers and Instrumentation course labs. Measure the capacitance of the capacitive sensor or the inductance of the inductive sensor with and without mine. You can use these values during your simulation. If you select an inductive sensor, it helps to pay attention to:
  - Start by designing a resonator in LTspice, and decide on an appropriate range for the oscillating resonance frequency.
  - Use the resonator, the LM358 and passive components to create an oscillator with an approximately square wave output. Note that the LM358 can only handle voltages between 0 and 5 V.
  - Double check that LTspice references the directory of the file lm358.mod correctly. If it doesn't, and doesn't give a warning, LTspice might use its default opamp model.

## Chapter 7

# Advanced FSM

During the DS-B course labs, you developed a FSM for the line follower. In Q4, for the EPO-2 project, you further complete the functionality of the robot by adding more components and functions (e.g., mine sensor, UART, etc). There are many different approaches for completing/extending your line follower's FSM for the EPO-2 project.

**Attention:** It is up to your group to decide which approach you use. Below we give two example approaches, but you are allowed to chose any other approaches as well. Hereby, we give you two (optional) example approaches,

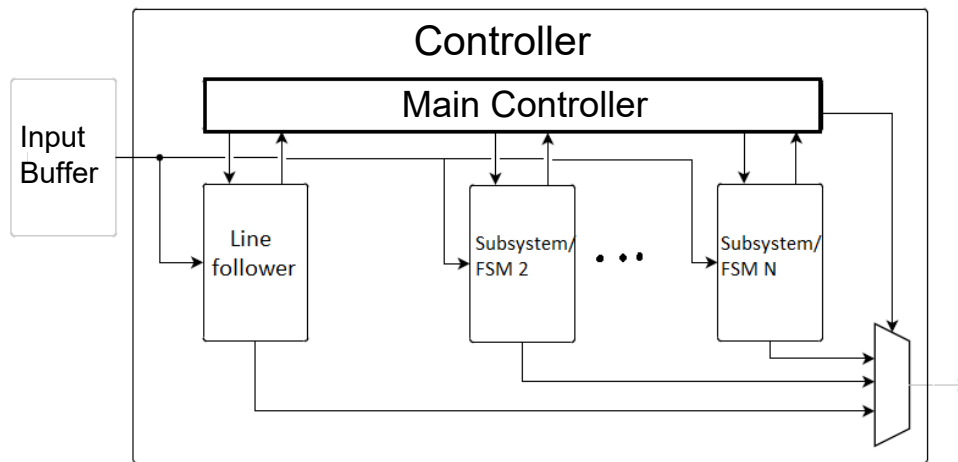
- **Example approach 1:** You can include additional states in your line follower's FSM and consequently, create a larger FSM.
- **Example approach 2:** You can create separate controllers/FSMs and then use an additional controller to select between these various controllers/FSMs. Figure 7.1 presents an example block diagram for this approach. In the controller block, the main controller is designed in such a way that it switches between various subsystems/FSMs using a multiplexer (MUX). The main controller is used to activate one subsystem/FSM (e.g., by keeping the other subsystems/FSMs in reset), and controlling the selection input of the mux.

### 7.1 Advantages and disadvantages

Both of the above two example approaches will work but there are advantages and disadvantages to both. Some remarks on the approach of example 1 (one single large FSM):

- Creating a single subsystem/FSM can involve less additional work;
- Adding too many states into a single controller may result in a too large and ambiguous design;
- Simulating a single large controller can be difficult and become very time consuming;





**Figure 7.1:** Example approach for the controller including various subsystems/FSMs using a multiplexer (MUX).

Some remarks on the approach of example 2 (separate subsystems/FSMs using MUX):

- Using separate subsystems/FSMs and a mux to select the proper set of outputs gives a more generic and extensible solutions;
- For the implementation, it may initially require additional work: you need to implement an additional controller and a mux;
- Extending the design with more subsystems/FSMs is simpler and quicker;
- The design can easily and quickly be simulated and tested in parts.

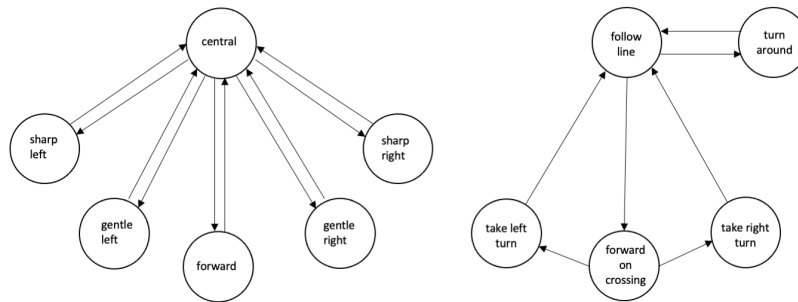
## 7.2 More thoughts on the advanced FSM for the robot

As mentioned, there are different ways to implement the FSM for the robot. Hereby, another thought on this is presented. Again, it is up to you how to do the actual implementation.

Below in Figure 7.2, it is shown how the FSM for the robot, instead of one large FSM, consists of 2 smaller FSMs that communicate with each other.

The FSM on the left is the line-follower that takes as input the signal from the timebase, the signals from the light sensors and the state of the FSM on the right. The outputs of this FSM control the motors.

The FSM on the right models the "current activity" of the robot. It takes as input the commands from the UART connection, the mine sensor and the light sensors. The outputs of this FSM are its state (used as input for the FSM on left) and the commands that are send to the UART connection. Especially the FSM on the right may need several more states to fully implement the behaviour as required for the robot.



**Figure 7.2:** Example FSM for the robot, instead of one large FSM, consisting of 2 smaller FSMs that communicate with each other.

# Bibliography

- [1] S. Brown, Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design, Third Edition*, McGraw-Hill, 2009
- [2] P. J. Ashenden, *The student's guide to VHDL, Second Edition*, Morgan Kaufmann Publishers, Inc, San Francisco, 2008
- [3] *Datasheet LM1086*, National Semiconductor, Online: <http://www.national.com/mpf/LM/LM1086.html>
- [4] B. Jacobs, X. van Rijnsoever, A.J. van Genderen, *Practicum Handleiding Programmeren in C*, TU Delft, The Netherlands
- [5] *Dijkstra's kortste pad algoritme* Online: <http://nl.wikipedia.org/wiki/Kortstepadalgoritme>
- [6] *Lee algoritme* Online: [http://en.wikipedia.org/wiki/Lee\\_algorithm](http://en.wikipedia.org/wiki/Lee_algorithm)
- [7] R. Bayer, *Windows serial Port Programming*, 2008, Online: <http://www.robbayer.com/files/serial-win.pdf>
- [8] *Serial Communications in Win32*, Microsoft, Online: <http://msdn.microsoft.com/en-us/library/ms810467.aspx>
- [9] *Datasheet LM1086*, National Semiconductor, Online: <http://www.national.com/mpf/LM/LM1086.html>
- [10] J. David Irwin, R. Mark Nelms, *Basic engineering circuit analysis*, Wiley, 2008.
- [11] *Datasheet LF155/LF156/LF256/LF257/LF355/LF356/LF357 JFET Input Operational Amplifiers*, National Semiconductor, Online: <http://www.national.com/ds/LF/LF155.pdf>.
- [12] *Datasheet LM741 Operation Amplifier*, National Semiconductor, Online: <http://www.national.com/ds/LM/LM741.pdf>.
- [13] *Shortest path problem*, Online: [http://en.wikipedia.org/wiki/Shortest\\_path\\_problem](http://en.wikipedia.org/wiki/Shortest_path_problem)
- [14] C.Y. Lee, *An Algorithm for Path Connections and Its Applications*, IRE Transactions on Electronic Computers, EC-10 (2), 1961: pp. 346-365
- [15] *Maze router: Lee algorithm*, Online: <http://www.eecs.northwestern.edu/haizhou/357/lec6.pdf>
- [16] R. Elling, B. Andeweg, J. de Jong, and C. Swankhuisen, *Writing for Readers with Little Time*. Noordhoff Uitgevers, 2012.

- [17] IEEE, “IEEE Reference Guide,” Last visited: May 2020. [Online].  
<https://ieeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf>
- [18] R. Grit, *Project Management - Projectmatig werken in de praktijk*. Noordhoff Uitgevers B.V., 2015.

## Appendix A

# Tutorial for ISE, de Xilinx design environment

*This section is written in English as a preparation of the complete manual translation.*

### Learning Objectives

During this session you will:

- get acquainted with the BASYS2 Xilinx Spartan3E-250 CP132 FPGA board
- get acquainted with the Xilinx ISE software pack

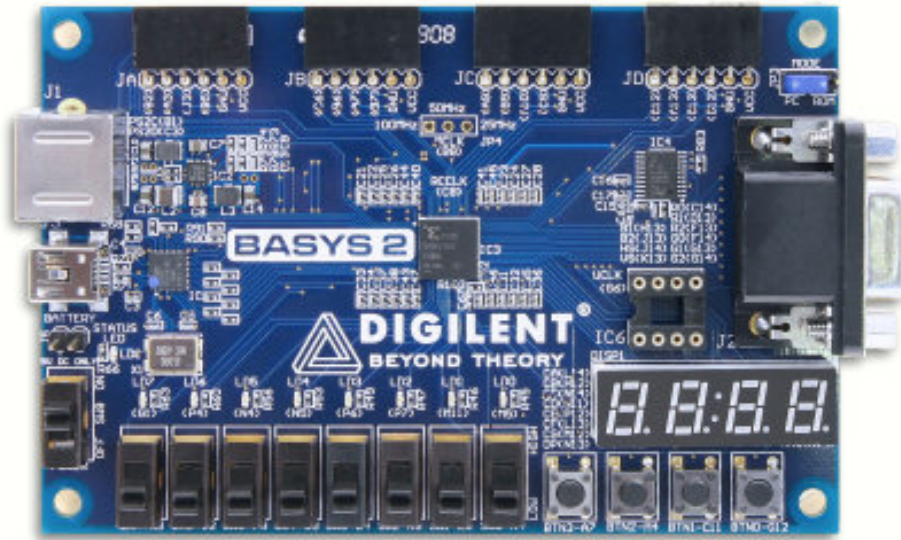
### A.1 Introduction

In this chapter you are going to follow a tutorial for using the Xilinx FPGA board and the associated software, Xilinx ISE. The most important thing is that you become familiar with the hardware and software used. This is done by the implementation of a 2-bit counter on the FPGA board. We assume that a VHDL description of such a counter is given (for example as a result of task 2.2). We assume that it is located in file `counter.vhdl`, that the name of the entity count is, and that the following inputs and outputs are present:

Port	Richting	Type
clk	IN	STD_LOGIC
reset	IN	STD_LOGIC
enable	IN	STD_LOGIC
count	OUT	STD_LOGIC_VECTOR (1 DOWNT0 0)

### A.2 Hardware: het Xilinx Spartan3E FPGA development board

First we will give a brief overview of the FPGA development board.



**Figure A.1:** Top view of the Digilent BASYS2 Xilinx Spartan3E FPGA development board

The Xilinx Spartan3E FPGA development board is, as the name implies, a development board. A development board is a Printed Circuit Board (PCB) that is organized around a particular chip such, for example, as a microprocessor or an FPGA. Furthermore there are a variety of components that make the use of the chip more easily. Some of these are necessary for the operation of the chip, such as the components that take care of the power supply, the clock and programming of the chip. Other components make the integration of the chip in a system easier. Consider, for example methods of communication (such as RS232 or CAN), a PS/2 port for a mouse or keyboard, or a VGA connector for a monitor.<sup>1</sup> Development boards are widely used in the industry for building prototypes.

An image of the board is shown in figure A.1.

The Digilent BASYS2 FPGA development board is designed around a Spartan3E FPGA chip. You see, among other switches, push buttons, LEDs and 7-segment displays. These are simply inputs and outputs you can use to test your design. Furthermore, you can see the expansion connectors. Through these connectors, the chip can communicate with the outside world. You also see a mini-USB connector. This connector is used to program the chip.

<sup>1</sup>If you are not familiar with RS232, CAN or PS/2 it is not a problem, these are just examples.

### A.3 Software: Xilinx ISE

Trying it yourself is the best way to learn something, it applies to (almost) everything. Therefore we now go through an example where all the steps that are important when working with Xilinx ISE are discussed. Take the time to go through all the steps! Later during the EPO-2 project you will have to work a lot with ISE and it is useful if you know how the software works.

Start the Xilinx ISE 14.6 Project Manager on a Windows PC (Start -> Programs -> Engineering -> Xilinx Design Tools -> ISE Design Suite 14.6 -> ISE Design Tools -> 64-bit Project Navigator) to start this tutorial.



#### Tab size

In the files you get, the default tab size is held by 8. Set, for a correct display, the editor on this tab size.

#### Making a new project

To create a design in Xilinx ISE, you must first create a new project. The project that you are going to make is called `counter`. For the project `counter`, you have to take the following steps:

1. Open in the menu File the option New Project.

This options opens a window with the title New Project Wizard.

2. Take the values from the table below and click on Next.

Name	counter
Location	path where <code>counter.vhdl</code> is stored
Working Directory	idem
Description	A 2-bits counter.
Top-level source type	HDL

3. Take the values from the table below and click on Next. Please note that the correct value for Device is the value that is printed on the chip itself. So check what chip it is on the Xilinx Spartan-3 FPGA board and fill in the value in device.

Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S250E
Package	CP132
Speed	-4
Top-level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE Mixed
Preferred Language	VHDL
Property Specification in Project File	Store non-default values only
Manual Compile Order	False
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	False

- Click on Finish to make the project.



#### Adjusting Design Properties

You can adjust the selected device (and most other options too) at the menu Project to open the option Design Properties.



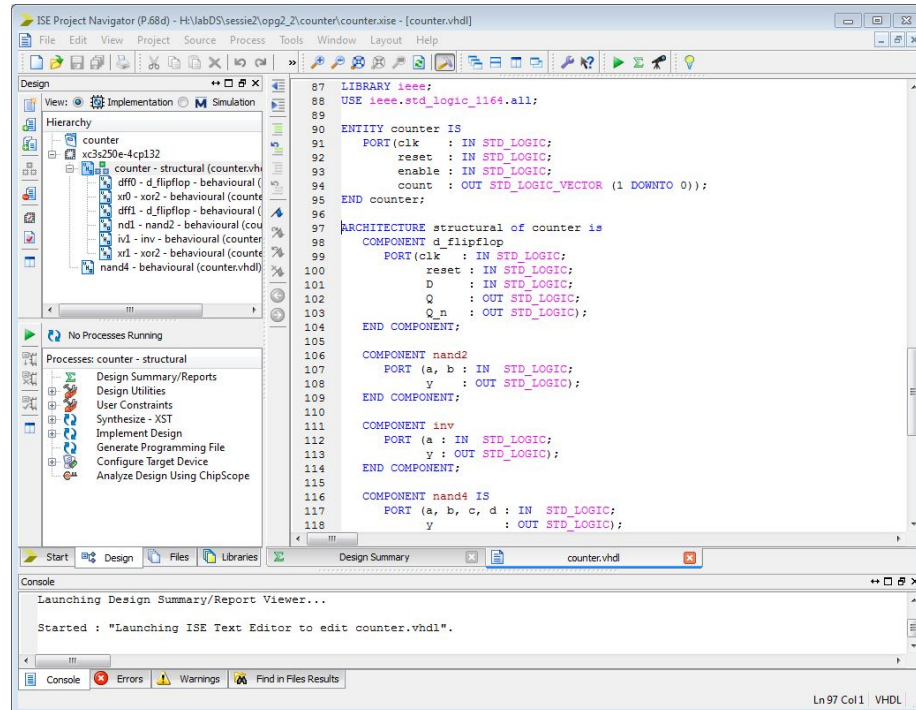
#### Do not use de Source Generator from Xilinx!

Source files made by the Xilinx Source Generator can use a number of libraries that contain errors. So do not use this method! In this tutorial, we describe the best way to create source files: either beforehand in an external editor, or later with the editor of the Xilinx ISE itself.

- Select in the menu Project the option Add Source and add the file `counter.vhdl` to the project.
- The contents of the file, in which the counter is located, can now be viewed by 2 times left-clicking on the name counter in the Hierarchy window (Top left).



## De Xilinx ISE user interface



You can see now that ISE consists of three frames: Console (below), Design (above that on the left with sub-frames Hierarchy and Processes), and the editor frames (top right and the largest frame).

The bottom frame is the Console. Xilinx ISE uses this screen to communicate with the user. Error messages and warnings are displayed here. Errors have red icons, warnings yellow.

The frame top left, titled Design, consists of two sub-frames. In the frame Hierarchy you see information about the project, including the name, the target device and the files in the project. By selecting a file, you can see in the frame Processes below what you can do with it.

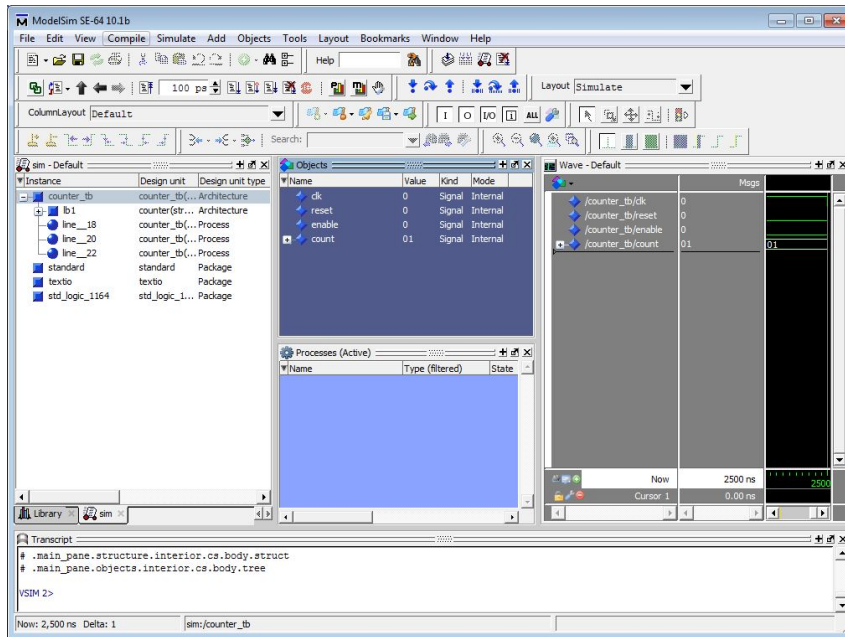
The largest and last frame, in the top right, is the editor frame. Here you can create source files or adjust them, like you can do with Notepad. The editor has line numbers and syntax highlighting.


## Simulating a design with Modelsim

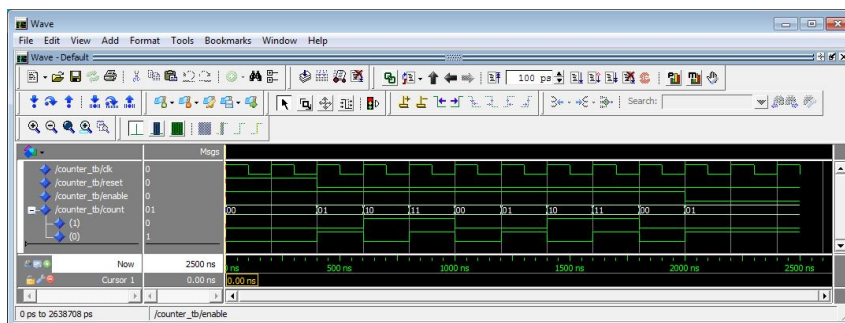
From Xilinx ISE it is also possible to simulate the design with Modelsim. This can be done in the following manner.

1. First, we add a testbench to the project. We assume that it is given in the file `counter_tb.vhdl`. Right click in the frame Hierarchy and select the option Add Source to add this file.
2. In the Design frame, just above the Hierarchy frame, select in the View option Simulation in stead of Implementation.
3. In the Hierarchy frame, select `counter_tb`.

4. In the Processes frame, click on the plus sign in Modelsim Simulator to make Simulate Behavioral Model visible.
5. Right-click on Simulate Behavioral Model and select in the pop-up menu Process Properties. In the appearing window specify for Simulation Run Time the value 2500ns and click OK.
6. Double-click on Simulate Behavioral Model. The ModelSim Simulator starts now. Compiling of the VHDL files is done automatically and a simulation is done for 2500 ns. If you've done everything correctly it will look like this:



7. Undock and maximize the Wave frame by clicking in the top right of the frame on the correct button. Then right-click in the new window and press  to view the entire waveform.



8. The simulation output can be saved as a file with the menu File and then the options Export and Image.

### Pin Assignment

In the simulation you have seen what happens to the output *count* if you set signals to the inputs *clk*, *reset* and *enable*. In order to allow the circuit to the work on the FPGA, the input signals have to, in one way or another, be provided to the FPGA. And we have to find a way to view the output signals. We will do this by connecting the input signals with buttons on the FPGA board and the output signals with LEDs that are present on the FPGA board. This can be achieved by specifying a so-called pin assignment in a constraint file.

1. We will now focus on the implementation of the design on the FPGA board. In the Design frame, just above the Hierarchy frame, select therefore the option Implementation in stead of Simulation at the View window.
2. Then, in menu Project, click New Source.... In the appearing window select Implementation Constraint File and specify the file name counter.ucf. Then click Next and then Finish. The name of the file will be shown in the Hierarchy window under entity counter and in the Edit window a editor for the file counter.ucf will be started. If no editor is started automatically, left click on the name counter.ucf in the Hierarchy window. Then, in the Processes window below it, double click on Edit Contraints(Text).
3. Type the following text in the edit window for counter.ucf

```
NET "clk" LOC = G12;
NET "reset" LOC = P11;
NET "enable" LOC = L3;
NET "count[0]" LOC = M5;
NET "count[1]" LOC = M11;
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

The first 5 rules specify how the ports must be connected to the design with the pins of the FPGA. The pins of the FPGA are in turn connected to components on the board as described in appendix C. This way port *clk* is connected on the FPGA to pin G12, which in turn is connected to push button BTNo. Ports *reset* and *enable* are thus connected to slide switches SW0 and SW1 and ports *count[0]* and *count[1]* with LEDs LD0 and LD1. The following table summarizes this.

clk	G12	BTNo
reset	P11	SW0
enable	L3	SW1
count[0]	M5	LD0
count[1]	M11	LD1

The rule which puts the option `CLOCK_DEDICATED_ROUTE` for *clk* on FALSE is used here to indicate that ISE does not need, in this case, to use the standard clock lines on the FPGA for the clock signal.



Normally, the clock signal is connected to the FPGA pin B8, to which a 50 MHz crystal is connected. If we would use in this example the direct signal as a clock signal, the counter will be working too fast to see which values will run through: the LEDs will light up seemingly continuous (half-strength). That is why we have connected the clock signal to a push button to see how the counter goes through different values step-by-step. Normally, when pin B8 is used to connect the clock, the option `CLOCK_DEDICATED_ROUTE` does NOT have to be specified.

4. Save the ucf file and close the text editor.

### Configure target device

A VHDL design and a Pin Assignment is all you need to create a programming file for a FPGA. After you create a programming file, you can program the FPGA. This can be done in two ways:

1. by programming the design in the volatile memory of the FPGA
2. by programming the design in external non-volatile memory

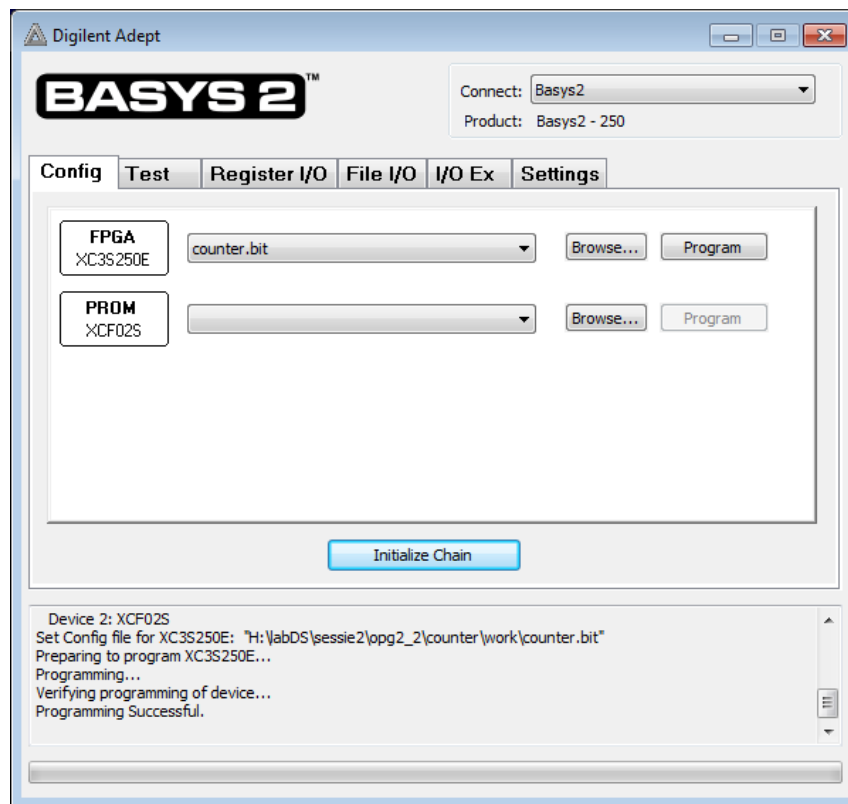
The first option is quite simple: The programming file you created, can be loaded directly into the FPGA memory and you have your design on the FPGA. However, if the voltage drops to zero on the FPGA, the volatile memory loses its information and you will have to reprogram it. By loading the programming file in an external non-volatile memory, you can remove the voltage from your FPGA without losing your programmed information.

To create a programming file you must ensure that you have a design without errors and a Pin Assignment. If your design contains errors, the synthesis stops of your design and the compiler says where the errors are. To create a programming file, you must do the following:

1. Make sure counter - structural is selected and then double-click on Generate Programming File. Please note that this could take a few minutes!

Now you have a programming file, you can load it into one of the memories on the FPGA board. This happens with Digilent Adept FPGA programming tool.

1. First switch on the BASYS2 board and connect, with a mini-USB cable, the BASYS2 board on to your PC.
2. Then open a separate software tool Digilent Adept.



3. Click on Browse... to select the file `counter.bit` in the work directory of your project. Next, click on Program. When you are asked about SCLK and JTAG CLK, click on OK.

After you have programmed the board, you can now try out the counter by switching off the switch SW<sub>0</sub> (*reset*), switching on SW<sub>1</sub> (*enable*), and repeatedly pressing the BTNo button (*clk*).

# Appendix B

## UART

### B.1 User constraint file uart.ucf

```
1 #=====
2 #   Pin assingment for BASYS2
3 #   Xilinx Spartan3E board
4 #=====
5
6 #=====
7 # clock and reset
8 #=====
9 NET "clk"      LOC = "B8" ;
10 NET "reset"    LOC = "A7"; #btn3 used for reset
11
12 #=====
13 # buttons & switches
14 #=====
15 # 2 push buttons
16 NET "write_data" LOC = "G12"; #btn<0>
17 NET "read_data"  LOC = "C11"; #btn<1>
18
19 # 8 slide switches
20 NET "sw<7>" LOC = "N3"; # Bank = 2, Signal name = SW7
21 NET "sw<6>" LOC = "E2"; # Bank = 3, Signal name = SW6
22 NET "sw<5>" LOC = "F3"; # Bank = 3, Signal name = SW5
23 NET "sw<4>" LOC = "G3"; # Bank = 3, Signal name = SW4
24 NET "sw<3>" LOC = "B4"; # Bank = 3, Signal name = SW3
25 NET "sw<2>" LOC = "K3"; # Bank = 3, Signal name = SW2
26 NET "sw<1>" LOC = "L3"; # Bank = 3, Signal name = SW1
27 NET "sw<0>" LOC = "P11"; # Bank = 2, Signal name = SW0
28
29
30 #=====
31 # 4-digit time-multiplexed 7-segment LED display
32 #=====
33 # Connected to Basys2 onBoard 7seg display
34 NET "seg<0>" LOC = "L14"; # Bank = 1, Signal name = CA
35 NET "seg<1>" LOC = "H12"; # Bank = 1, Signal name = CB
36 NET "seg<2>" LOC = "N14"; # Bank = 1, Signal name = CC
37 NET "seg<3>" LOC = "N11"; # Bank = 2, Signal name = CD
38 NET "seg<4>" LOC = "P12"; # Bank = 2, Signal name = CE
39 NET "seg<5>" LOC = "L13"; # Bank = 1, Signal name = CF
40 NET "seg<6>" LOC = "M12"; # Bank = 1, Signal name = CG
41 NET "seg<7>" LOC = "N13"; # Bank = 1, Signal name = DP
```

```

42 NET "an<3>" LOC = "K14"; # Bank = 1, Signal name = AN3
43 NET "an<2>" LOC = "M13"; # Bank = 1, Signal name = AN2
44 NET "an<1>" LOC = "J12"; # Bank = 1, Signal name = AN1
45 NET "an<0>" LOC = "F12"; # Bank = 1, Signal name = AN0
46
47 #=====
48 # 8 discrete led
49 #=====
50 NET "Led<7>" LOC = "G1" ; # Bank = 3, Signal name = LD7
51 NET "Led<6>" LOC = "P4" ; # Bank = 2, Signal name = LD6
52 NET "Led<5>" LOC = "N4" ; # Bank = 2, Signal name = LD5
53 NET "Led<4>" LOC = "N5" ; # Bank = 2, Signal name = LD4
54 NET "Led<3>" LOC = "P6" ; # Bank = 2, Signal name = LD3
55 NET "Led<2>" LOC = "P7" ; # Bank = 3, Signal name = LD2
56 NET "Led<1>" LOC = "M11" ; # Bank = 2, Signal name = LD1
57 NET "Led<0>" LOC = "M5" ; # Bank = 2, Signal name = LD0
58
59 # Pin assignment for PMOD connectors
60 #NET "JA<1>" LOC = "B2" ;
61 NET "sensor<2>" LOC = "A3" ;
62 NET "sensor<1>" LOC = "J3" ;
63 NET "sensor<0>" LOC = "B5" ;
64
65 #NET "JB<1>" LOC = "C6" ;
66 #NET "JB<2>" LOC = "B6" ;
67 #NET "JB<3>" LOC = "C5" ;
68 #NET "JB<4>" LOC = "B7" ;
69
70 NET "servo<1>" LOC = "A9" ; # servo left
71 NET "servo<0>" LOC = "B9" ; # servo right
72 NET "rx" LOC = "A10" ; #xbee out (received data)
73 NET "tx" LOC = "C9" ; #xbee in (data to send)
74
75 #NET "JD<1>" LOC = "C12" ;
76 #NET "JD<2>" LOC = "A13" ;
77 #NET "JD<3>" LOC = "C13" ;
78 #NET "JD<4>" LOC = "D12" ;
79

```

## B.2 Template C program to interface with Windows COM port

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <Windows.h>
4 #include <string.h>
5
6 #define COMPORT "COM5"
7 #define BAUDRATE CBR_9600
8
9 //-----
10 // Function: initSio
11 // Description: initializes the parameters as Baudrate, Bytesize,
12 //             Stopbits, Parity and Timeoutparameters of
13 //             the COM port
14 //-----
15 void initSio(HANDLE hSerial){
16
17     COMMTIMEOUTS timeouts = {0};
18     DCB dcbSerialParams = {0};

```

```

19     dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
20
21
22     if (!GetCommState(hSerial, &dcbSerialParams)) {
23         //error getting state
24         printf("error getting state \n");
25     }
26
27     dcbSerialParams.BaudRate = BAUDRATE;
28     dcbSerialParams.ByteSize = 8;
29     dcbSerialParams.StopBits = ONESTOPBIT;
30     dcbSerialParams.Parity = NOPARITY;
31
32     if (!SetCommState(hSerial, &dcbSerialParams)){
33         //error setting serial port state
34         printf("error setting state \n");
35     }
36
37     timeouts.ReadIntervalTimeout = 50;
38     timeouts.ReadTotalTimeoutConstant = 50;
39     timeouts.ReadTotalTimeoutMultiplier = 10;
40
41     timeouts.WriteTotalTimeoutConstant = 50;
42     timeouts.WriteTotalTimeoutMultiplier = 10;
43
44     if (!SetCommTimeouts(hSerial, &timeouts)){
45         //error occureed. Inform user
46         printf("error setting timeout state \n");
47     }
48 }
49
50 //-----
51 // Function: readByte
52 // Description: reads a single byte from the COM port into
53 //              buffer buffRead
54 //-----
55 int readByte(HANDLE hSerial, char *buffRead) {
56
57     DWORD dwBytesRead = 0;
58
59     if (!ReadFile(hSerial, buffRead, 1, &dwBytesRead, NULL))
60     {
61         printf("error reading byte from input buffer \n");
62     }
63     printf("Byte read from read buffer is: %c \n", buffRead[0]);
64     return(0);
65 }
66
67 //-----
68 // Function: writeByte
69 // Description: writes a single byte stored in buffRead to
70 //              the COM port
71 //-----
72 int writeByte(HANDLE hSerial, char *buffWrite){
73
74     DWORD dwBytesWritten = 0;
75
76     if (!WriteFile(hSerial, buffWrite, 1, &dwBytesWritten, NULL))
77     {
78         printf("error writing byte to output buffer \n");
79     }
80     printf("Byte written to write buffer is: %c \n", buffWrite[0]);

```



```

81     return(0);
82 }
83
84
85 int main()
86 {
87     HANDLE hSerial;
88
89
90     char byteBuffer[BUFSIZ+1];
91
92     //-----
93     // Open COMPORT for reading and writing
94     //-----
95     hSerial = CreateFile(COMPORT,
96         GENERIC_READ | GENERIC_WRITE,
97         0,
98         0,
99         OPEN_EXISTING,
100        FILE_ATTRIBUTE_NORMAL,
101        0
102    );
103
104    if(hSerial == INVALID_HANDLE_VALUE){
105        if(GetLastError() == ERROR_FILE_NOT_FOUND){
106            //serial port does not exist. Inform user.
107            printf(" serial port does not exist \n");
108        }
109        //some other error occurred. Inform user.
110        printf(" some other error occurred. Inform user.\n");
111    }
112
113    //-----
114    // Initialize the parameters of the COM port
115    //-----
116
117    initSio(hSerial);
118
119    while ( 1 ) {
120        gets(byteBuffer);
121
122        if (byteBuffer[0] == 'q') // end the loop by typing 'q'
123            break;
124
125        writeByte(hSerial, byteBuffer);
126        readByte(hSerial, byteBuffer);
127    }
128
129    printf("ZIGBEE IO DONE!\n");
130    return 0;
131
132    CloseHandle(hSerial);
133 }

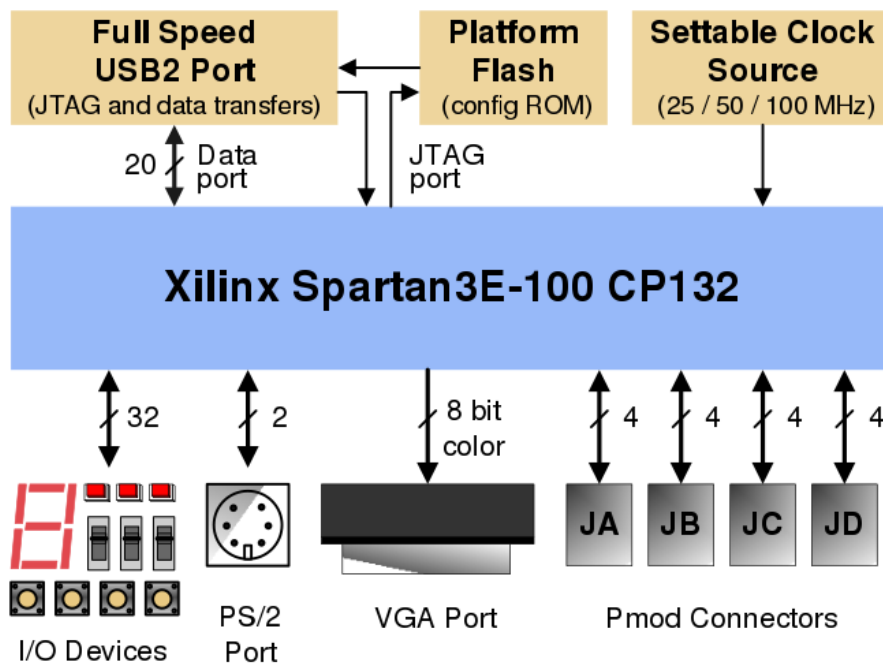
```

## Appendix C

# Overview of the FPGA pins

*This section is written in English as a preparation of the complete manual translation.*

The following picture shows the various components on the BASYS 2 FPGA board that are directly connected to FPGA pins.



The Spartan-3 FPGA is embedded in a so-called ball-grid package. The “pins” are disposed in an organized grid. The rows of the grid are addressed by letters from A to T, the columns with numbers from 1 to 16, where the top left pin is A1. This appendix provides a brief overview of the connections of the various components that are present on the BASYS 2 FPGA board, such as the LEDs, the 7-segment displays and the buttons. The full list of connections is shown at the end of this appendix.

## C.1 7-Segment displays

On the board four 7-segment displays are present. These displays share the same data lines, but are each provided with an independent enable. All connections of the displays are active-low, so in order to turn a segment on, a '0' must be written to the corresponding FPGA pin.

Moreover, the sharing of the same data lines among the displays, requires a scanning technique to be employed to set the segments of the different displays. These data and the enable of each display must be connected in succession, and then plotted. Note, that also the enable signals are active-low.

The displays are connected as follows:

Segment	Pin
A	L14
B	H12
C	N14
D	N11
E	P12
F	L13
G	M12
H	N13

Display enable	Pin
Enable_0	K14
Enable_1	M13
Enable_2	J12
Enable_3	F12

## C.2 Leds

On the board eight LEDs are present. The LEDs are active-high: setting a '1' to the corresponding FPGA pin will turn the LED on.

led	Pin
LED_0	M5
LED_1	M11
LED_2	P7
LED_3	P6
LED_4	N5
LED_5	N4
LED_6	P4
LED_7	G1

## C.3 Slide Switches

On the board eight slide switches are available. When the switch is pushed upwards, a '1' is provided to the corresponding FPGA pin.

Switch	Pin
SW_0	P11
SW_1	L3
SW_2	K3
SW_3	B4
SW_4	G3
SW_5	F3
SW_6	E2
SW_7	N3

## C.4 Push buttons

On the board four buttons are present. When the button is pressed, a '1' is provided to the corresponding FPGA pin.

Drukknop	Pin
BTN_0	G12
BTN_1	C11
BTN_2	M4
BTN_3	A7

## C.5 Oscillator

The board is equipped with a 50MHz oscillator which can be used as a clock signal.

Oscillator	Pin
oscillator	B8

## C.6 Full list of connections

```

1 # This file is a general .ucf for Basys2 rev C board
2 # To use it in a project:
3 # - remove or comment the lines corresponding to unused pins
4 # - rename the used signals according to the project
5
6 # clock pin for Basys2 Board
7 NET "mclk" LOC = "B8"; # Bank = 0, Signal name = MCLK
8 NET "uclk" LOC = "M6"; # Bank = 2, Signal name = UCLK
9 NET "mclk" CLOCK_DEDICATED_ROUTE = FALSE;
10 NET "uclk" CLOCK_DEDICATED_ROUTE = FALSE;
11
12 # Pin assignment for EppCtl
13 # Connected to Basys2 onBoard USB controller
14 NET "EppAstb" LOC = "F2"; # Bank = 3
15 NET "EppDstb" LOC = "F1"; # Bank = 3
16 NET "EppWR" LOC = "C2"; # Bank = 3
17
18 NET "EppWait" LOC = "D2"; # Bank = 3
19
20
21 NET "EppDB<0>" LOC = "N2"; # Bank = 2
22 NET "EppDB<1>" LOC = "M2"; # Bank = 2

```

```

23 NET "EppDB<2>" LOC = "M1"; # Bank = 3
24 NET "EppDB<3>" LOC = "L1"; # Bank = 3
25 NET "EppDB<4>" LOC = "L2"; # Bank = 3
26 NET "EppDB<5>" LOC = "H2"; # Bank = 3
27 NET "EppDB<6>" LOC = "H1"; # Bank = 3
28 NET "EppDB<7>" LOC = "H3"; # Bank = 3
29
30
31 # Pin assignment for DispCtl
32 # Connected to Basys2 onBoard 7seg display
33 NET "seg<0>" LOC = "L14"; # Bank = 1, Signal name = CA
34 NET "seg<1>" LOC = "H12"; # Bank = 1, Signal name = CB
35 NET "seg<2>" LOC = "N14"; # Bank = 1, Signal name = CC
36 NET "seg<3>" LOC = "N11"; # Bank = 2, Signal name = CD
37 NET "seg<4>" LOC = "P12"; # Bank = 2, Signal name = CE
38 NET "seg<5>" LOC = "L13"; # Bank = 1, Signal name = CF
39 NET "seg<6>" LOC = "M12"; # Bank = 1, Signal name = CG
40 NET "dp" LOC = "N13"; # Bank = 1, Signal name = DP
41
42 NET "an<3>" LOC = "K14"; # Bank = 1, Signal name = AN3
43 NET "an<2>" LOC = "M13"; # Bank = 1, Signal name = AN2
44 NET "an<1>" LOC = "J12"; # Bank = 1, Signal name = AN1
45 NET "an<0>" LOC = "F12"; # Bank = 1, Signal name = AN0
46
47 # Pin assignment for LEDs
48 NET "Led<7>" LOC = "G1"; # Bank = 3, Signal name = LD7
49 NET "Led<6>" LOC = "P4"; # Bank = 2, Signal name = LD6
50 NET "Led<5>" LOC = "N4"; # Bank = 2, Signal name = LD5
51 NET "Led<4>" LOC = "N5"; # Bank = 2, Signal name = LD4
52 NET "Led<3>" LOC = "P6"; # Bank = 2, Signal name = LD3
53 NET "Led<2>" LOC = "P7"; # Bank = 3, Signal name = LD2
54 NET "Led<1>" LOC = "M11"; # Bank = 2, Signal name = LD1
55 NET "Led<0>" LOC = "M5"; # Bank = 2, Signal name = LD0
56
57 # Pin assignment for SWs
58 NET "sw<7>" LOC = "N3"; # Bank = 2, Signal name = SW7
59 NET "sw<6>" LOC = "E2"; # Bank = 3, Signal name = SW6
60 NET "sw<5>" LOC = "F3"; # Bank = 3, Signal name = SW5
61 NET "sw<4>" LOC = "G3"; # Bank = 3, Signal name = SW4
62 NET "sw<3>" LOC = "B4"; # Bank = 3, Signal name = SW3
63 NET "sw<2>" LOC = "K3"; # Bank = 3, Signal name = SW2
64 NET "sw<1>" LOC = "L3"; # Bank = 3, Signal name = SW1
65 NET "sw<0>" LOC = "P11"; # Bank = 2, Signal name = SW0
66
67 NET "btn<3>" LOC = "A7"; # Bank = 1, Signal name = BTN3
68 NET "btn<2>" LOC = "M4"; # Bank = 0, Signal name = BTN2
69 NET "btn<1>" LOC = "C11"; # Bank = 2, Signal name = BTN1
70 NET "btn<0>" LOC = "G12"; # Bank = 0, Signal name = BTN0
71
72 # Loop back/demo signals
73 # Pin assignment for PS2
74 NET "PS2C" LOC = "B1" | DRIVE = 2 | PULLUP; # Bank = 3, Signal
  name = PS2C
75 NET "PS2D" LOC = "C3" | DRIVE = 2 | PULLUP; # Bank = 3, Signal
  name = PS2D
76
77 # Pin assignment for VGA
78 NET "HSYNC" LOC = "J14" | DRIVE = 2 | PULLUP; # Bank = 1, Signal
  name = HSYNC
79 NET "VSYNC" LOC = "K13" | DRIVE = 2 | PULLUP; # Bank = 1, Signal
  name = VSYNC
80

```

```

81 NET "OutRed<2>" LOC = "F13" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = RED2
82 NET "OutRed<1>" LOC = "D13" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = RED1
83 NET "OutRed<0>" LOC = "C14" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = RED0
84 NET "OutGreen<2>" LOC = "G14" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = GRN2
85 NET "OutGreen<1>" LOC = "G13" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = GRN1
86 NET "OutGreen<0>" LOC = "F14" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = GRN0
87 NET "OutBlue<2>" LOC = "J13" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = BLU2
88 NET "OutBlue<1>" LOC = "H13" | DRIVE = 2 | PULLUP ; # Bank = 1,
    Signal name = BLU1
89
90 # Loop Back only tested signals
91 NET "PIO<72>" LOC = "B2" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JA1
92 NET "PIO<73>" LOC = "A3" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JA2
93 NET "PIO<74>" LOC = "J3" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JA3
94 NET "PIO<75>" LOC = "B5" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JA4
95
96 NET "PIO<76>" LOC = "C6" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JB1
97 NET "PIO<77>" LOC = "B6" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JB2
98 NET "PIO<78>" LOC = "C5" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JB3
99 NET "PIO<79>" LOC = "B7" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JB4
100
101 NET "PIO<80>" LOC = "A9" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JC1
102 NET "PIO<81>" LOC = "B9" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JC2
103 NET "PIO<82>" LOC = "A10" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JC3
104 NET "PIO<83>" LOC = "C9" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JC4
105
106 NET "PIO<84>" LOC = "C12" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JD1
107 NET "PIO<85>" LOC = "A13" | DRIVE = 2 | PULLUP ; # Bank = 2, Signal
    name = JD2
108 NET "PIO<86>" LOC = "C13" | DRIVE = 2 | PULLUP ; # Bank = 1, Signal
    name = JD3
109 NET "PIO<87>" LOC = "D12" | DRIVE = 2 | PULLUP ; # Bank = 2, Signal
    name = JD4

```

