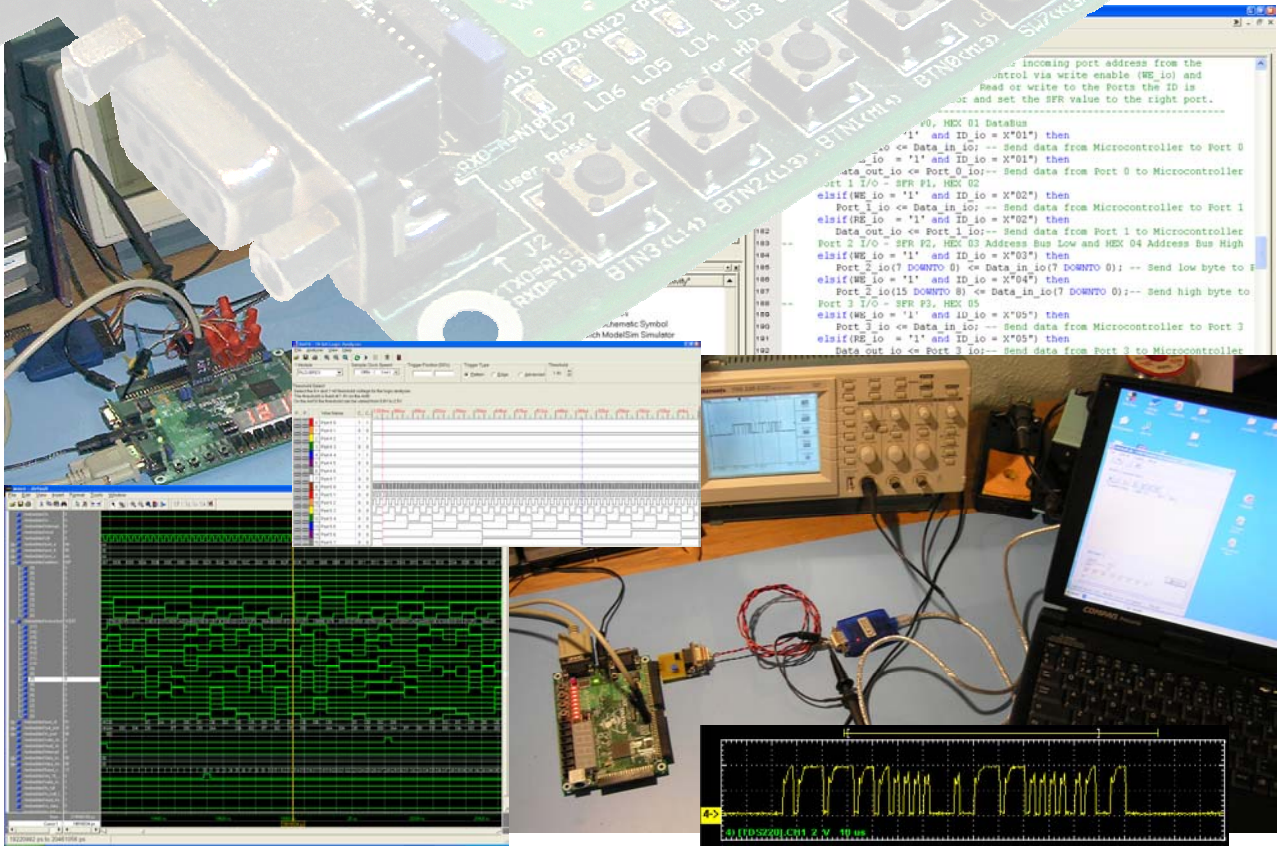
Honours Project

FPGA Implementation of a Simple Microprocessor

Napier University Edinburgh



BGEPB1 MicroController for Spartan-3 PFGA

BG ELEKTRONIK
Hardware & Software Development

Title Page

NAME: **Benjamin Grydehoej**

MATRICULATION NO.: **04007714**

UNIVERSITY: **Napier University Edinburgh**

EDUCATION: **BEng (Honours) Electronic and Computer Engineering**

MODULE TITLE: **BEng Honours Project**

MODULE NO.: **SE42201**

PROJECT TITLE: **FPGA Implementation of a Simple Microprocessor**

SUPERVISOR: **Dr. Thomas David Binnie**

SUBMISSION DATE: **5/5/2006**

Abstract

This report covers implementation of a simple 8-bit microprocessor in a FPGA, the design is made as an emulated standard 8051 microcontroller. It is build based on the free PicoBlaze™ IP Core from Xilinx, containing a Special Function Register which is specifically chosen for this microcontroller. BGEPB1 is short for (BG-Electronic PicoBlaze version 1). The microcontroller is implemented with parallels input and output ports (I/O ports), serial UART, timers and interrupts. The microcontroller interface is programmed in VHDL and the test programs for the microcontroller are made in C-code language using the PCCOMP compiler by Francesco Poderico. Tested in both ModelSim, a simulating tool, and in practical on the development board named Spartan-3 starter kit from Xilinx using the FPGA XC3S200. The second part of the report is regarding the design of a CAN bus hardware interface for the development board and a CAN VHDL interface for transmitting data through the CAN bus level converter out on to the CAN bus. The last part of the report is a setup guide for the software used to implemented and design new VHDL function.

Acknowledgement

I thank the following individuals for their contribution of PicoBlaze microcontroller Core and Development tool there are used for this project.

- **Ken Chapman, Xilinx Ltd. Benchmark House**
  PicoBlaze™ core and Serial UART
- **Xilinx Inc.**
  ISE Service Pack
- **Model Technology, a Mentor Graphics Corporation**
  Model*Sim* XE II/Starter
- **Francesco Poderico**
  PCCOMP PicoBlaze C Compiler

Table of Contents

Chapter 1

Introduction

1.1. About the project

A microcontroller in a Field-Programmable Gate Array (FPGA) is not world news, but a free 8051 emulate core in a FPGA is not available on the marked at the moment. There is an embedded microcontroller core on the market at the moment which matches the project, though without Control Area Network (CAN) bus Interface. It is the PB8051 Xilinx AllianceCORE™**[1]** to the price of $ 495.95.

The embedded microprocessor cores for FPGA is split up in Hard-core and Soft-core processors, a Hard-core Processor is the IBM PowerPC™ 405 32-Bit RISC processor which run on Xilinx Virtex-II Pro and Virtex-4. The Soft-core processor is a MicroBlaze™ 32-bit RISC core which runs up to 180MHz in a Virtex-4 with 166 MIPS build for complex systems, networking, telecommunication, data communication and embedded systems. All these microprocessor cores need a license to be used in a product. Another free soft-core processor from Xilinx is the PicoBlaze™ core which is an 8-Bit RISC processor this can be implemented on Virtex™ and Spartan™ series of FPGAs and CoolRunner™-II CPLDs. This microprocessor is the one chosen for this project because it is free and makes it possible to run in a low cost Spartan 3 FPGA. The purpose of this project is to make a cheap microcontroller core with peripherals like an 8051 standard microcontroller plus a CAN bus interface that makes it possible to customize the core for special projects.

1.2. Aim of the project

The aim for this project is to get know-how about FPGA and Very High Spe/ed Integrated Circuit Hardware Description Language (VHDL), and to integrated PicoBlaze™ processor in the Spartan-3 FPGA with Input and Output for parallel and serial interfaces and finally simulate and test the project in practical.

1.3. Project description

The block diagram in figure 1.1 shows the upcoming design of an emulated 8051 microcontroller, consisting of the Xilinx PicoBlaze™ microprocessor, with a instruction Read Only Memory (ROM) which makes it possible to run machine code from the ROM, generate by assembly or C code compiler. The machine code for the Instruction ROM is uploaded via the Xilinx program called Project navigator, using the iMPACT tool. The machine code is uploaded with the VHDL code for the project via Joint Test Action Group (JTAG).

The size of the Instruction ROM is only 1K x 16 and very small and will only be used as a Boot or Monitor ROM with all necessary information for communication to the peripherals, for more external ROM space available in the serial Flash which communicates via serial data control by the Serial Flash ROM interface Block.

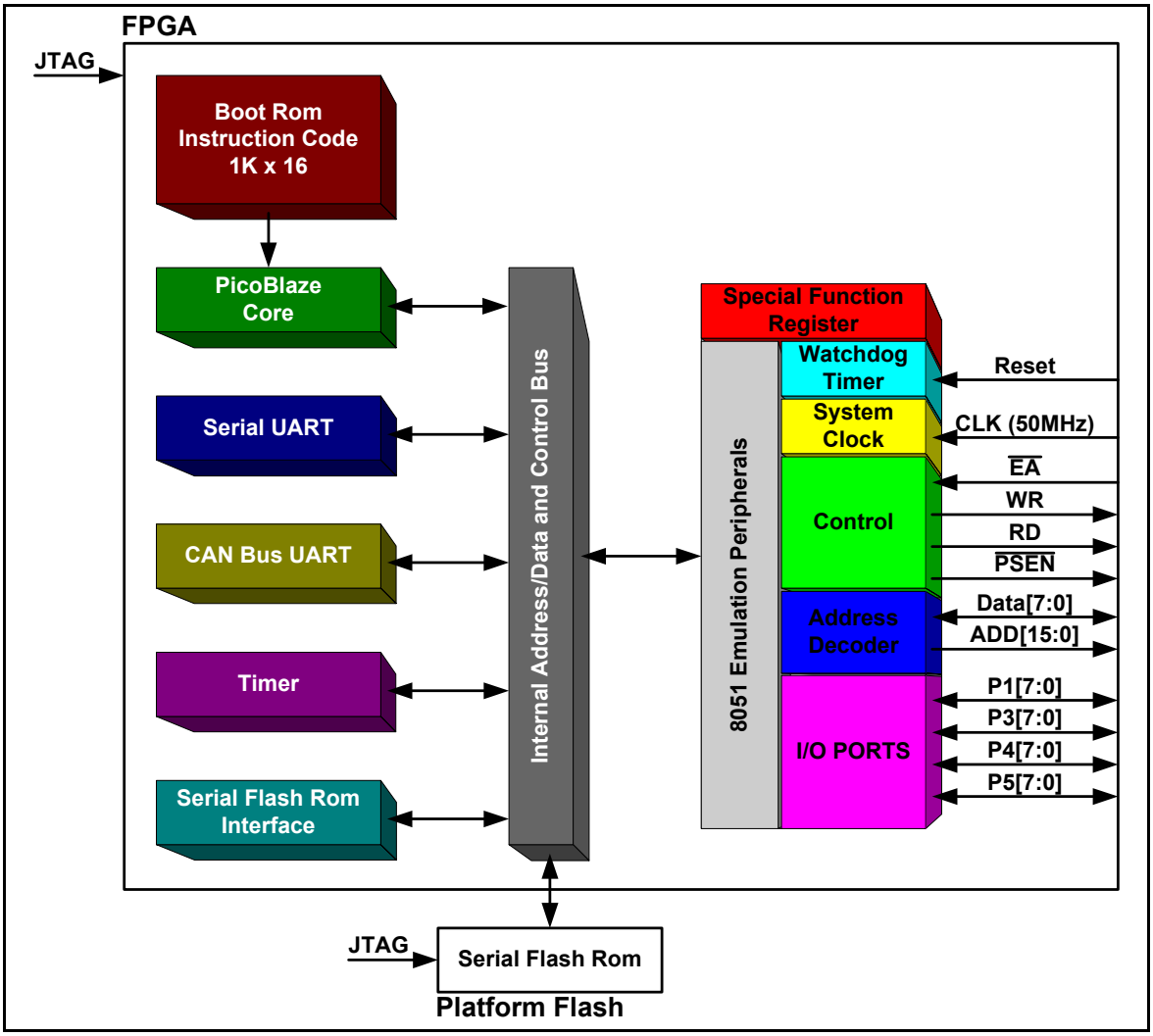


Figure 1.1 – Block diagram over BGEPB1 Emulated 8051 Microcontroller

1.4. Requirement specification

The specification for the project is described in this paragraph and all the special function calls are made out from the list for Special Function Register (SFR) showed in table 1.1. The project consists of building an embedded microcontroller in a FPGA with a CAN-Bus interface shown in the block diagram at page 8 figure 1.1. The specification of the project is listed in bullets point under this text.

- **PICOBLAZE™ CORE AND BOOT ROM** (Use the PicoBlaze™ features showed on page 13)
- **SERIAL UART (RS232)** (Standard configuration 1 start bit, 8 data bit, no parity and 1 stop bit)
- **CAN BUS UART** (Designed to ISO 11898-1, CAN 2.0A & B, support bit rates up to 1Mbit/s)
- **TIMER** (Timer 0 as 8-bit Timer and Timer 1 as 16-bit Timer)
- **SERIAL FLASH ROM INTERFACE** (Controller interface or extern serial program store up to 2Mbit)
- **SPECIAL FUNCTION REGISTER (SFR)** (The SFR control all the call to ports, serial UART, Timer etc.)
- **WATCHDOG TIMER** (Automatics reset of the microcontroller with problems in the code)
- **SYSTEM CLOCK** (Standard option is 50MHz, run up to 200MHz or 100MIPS in a Virtex-II Pro FPGA)
- **CONTROL** (Control signal for external Rom and RAM or other peripherals components)
- **ADDRESS DECODER** (Address bus expander up to 16 bit wide)
- **I/O PORTS** (Port 0, 1, 3, 4 and 4 with external interrupts and Serial RS232 and CAN interface)

Symbol:	Name:	Address:
P0	Port 0	HEX 01
P1	Port 1	HEX 02
P2L	Port 2 (Address Bus low byte "the lower 8-bit part of 16-bit")	HEX 03
P2H	Port 2 (Address Bus high byte "the higher 8-bit part of 16-bit")	HEX 04
P3	Port 3	HEX 05
P4	Port 4	HEX 06
P5	Port 5	HEX 07
SBUF	Serial channel buffer register	HEX 08
TLBS	Timer Low BAUE Rate Serial (Low byte part of 16-bit)	HEX 09
THBS	Timer High BAUE Rate Serial (High byte part of 16-bit)	HEX 0A
SCON	Serial channel control register	HEX 0B
IEN0	Interrupt enable register 0	HEX 0C
IEN1	Interrupt enable register 1	HEX 0D
ISC0	Interrupt service control register	HEX 0E
TCON	Timer service control register	HEX 0F
TC0	Timer Count 0 (8-bit)	HEX 10
TCL1	Timer Count Low 1 (part of 16-bit)	HEX 11
TCH1	Timer Count Low 1 (part of 16-bit)	HEX 12

Table 1.1 – List over Special Function Register.

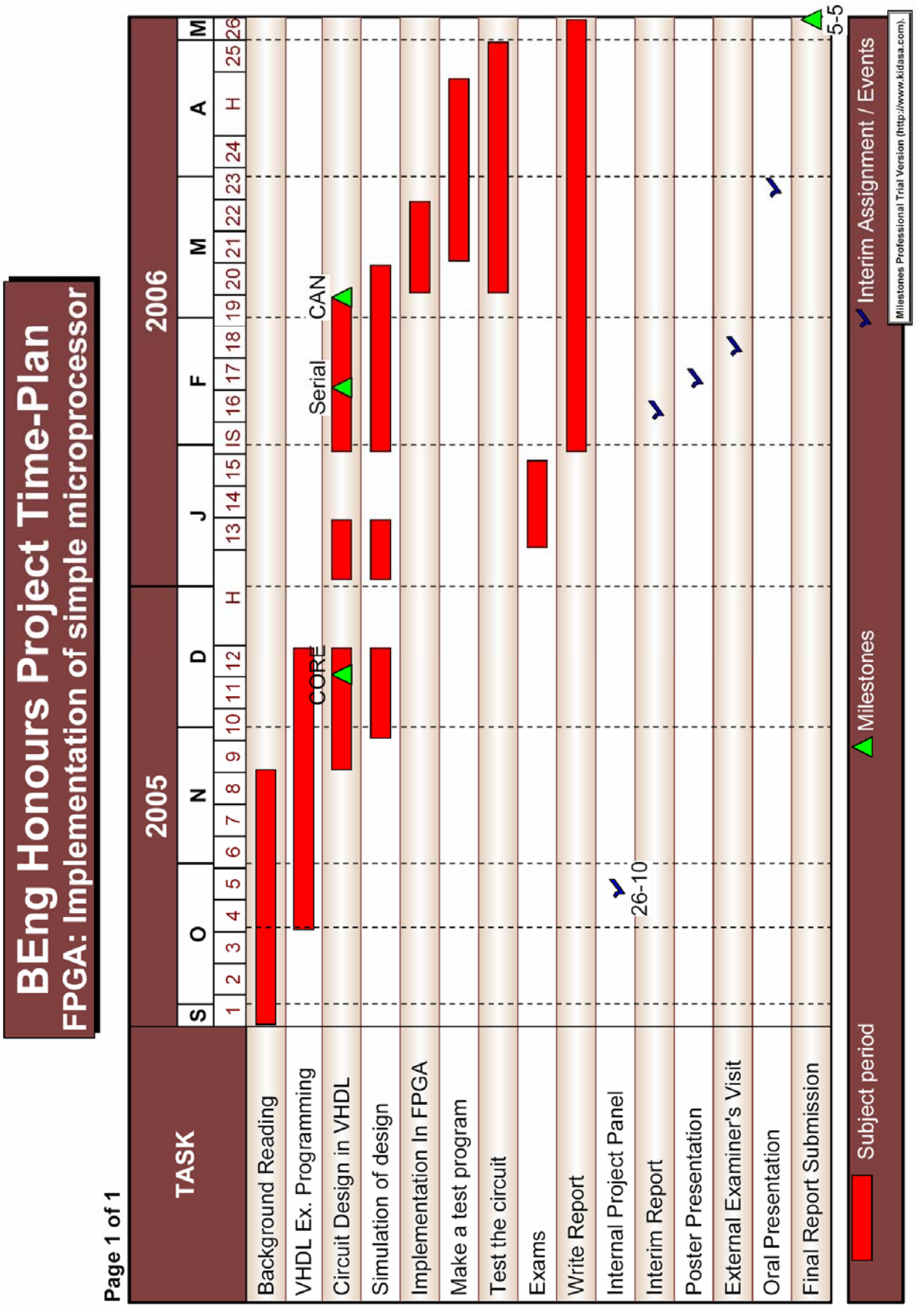
1.5. Problem solution

The task concerns the building an embedded microcontroller in a FPGA with a CAN bus interface from the requirement specifications at page 9. The microprocessor used for this project is the Xilinx PicoBlaze™ microprocessor core and the task is to implement parallel Input/Output port interfaces with Interrupts, serial UART, Timer and a CAN BUS interface. It can be necessary to implement the VHDL code giving access to the Serial Flash ROM for more program space.

The PicoBlaze™ core, the Instruction ROM and the serial UART is VHDL code which will be downloaded as free IP Core available from Xilinx.com homepage. The rest of the blocks in the block diagram in figure 1.1 at page 8, are functions of VHDL code constructed from scats.

The process for the project will be implementation of the PicoBlaze™ core with Boot ROM and Serial UART and create and implemented an Input/output interface with Interrupt control. Two different timers, a Timer 0 using an 8-bit counter and a Timer 1 which uses a 16-bit counter. The last unit there will be create and implemented is the CAN bus UART which also will be build from nothing. All the functions will be controlled by the Special Function Register showed in table 1.1 at page 9.

1.6 Time-Plan


BEng Honours Project Time-Plan
FPGA: Implementation of simple microprocessor
Page 1 of 1
TASK
2005
2006
S O N D J F M A M
1 2 3 4 5 6 7 8 9 10 11 12 H 13 14 15 IS 16 17 18 19 20 21 22 23 24 H 25 26
Background Reading
VHDL Ex. Programming
Circuit Design in VHDL
Simulation of design
Implementation In FPGA
Make a test program
Test the circuit
Exams
Write Report
Internal Project Panel
Interim Report
Poster Presentation
External Examiner's Visit
Oral Presentation
Final Report Submission
CORE
Serial
CAN
26-10
5-5
Subject period
Milestones
Interim Assignment / Events
Milestones Professional Trial Version (http://www.kidasa.com).

Chapter 2

Implementation of PicoBlaze™ with I/O ports interface

2.1. Introduction

This chapter descripts the PicoBlaze core and its features for the processor and how to implement the microcontroller core in a Spartan-3 FPGA with parallel Inputs and Outputs and interrupt service controller for external interrupt at I/O pins. This chapter will cover all the steps from the design of I/O ports VHDL code and set the Xilinx project navigator up and make a C language test program for the I/O ports to test the system in hardware.

2.2. Background of PicoBlaze™

The PicoBlaze microcontroller is a compact core, making it possible to download free version without IP license from Xilinx.com after registration of user. The microcontroller is an embedded 8-bit RISC core optimized for the Spartan-3, Virtex-II, and Virtex-II Pro FPGA families. The PicoBlaze microcontroller is optimized for efficiency and low development cost. It occupies just 96 FPGA slices, or 12,5% of an XC3S50 FPGA, and performs a respectable 44 to 100 million instructions per second (MIPS). For development on the PicoBlaze microcontroller the tool named Xilinx project navigator version 6.3.03 is used. This is a free software from Xilinx ready to download at Xilinx.com and makes it possible to add I/O ports, serial UART, timer, etc. To make C language test software for the microprocessor there are used two compilers one from Francesco Poderico's named PCCOMP, a DOS version, which compile the C language code to ASM code written in notepad. The second compiler is from Xilinx and named KCPSM3 which compile the ASM code to VHDL and making it ready to download to the FPGA after complete compiling of the project in the Xilinx project navigator.

Features:

The block diagram in figure 2.1 show the PicoBlaze microcontrollers supports the following features.[2]

- 16 byte-wide general-purpose data registers
- 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration
- Byte-wide Arithmetic Logic Unit (ALU) with CARRY and ZERO indicator flags
- 64-byte internal scratchpad RAM
- 256 input and 256 output ports for easy expansion and enhancement
- Automatic 31-location CALL/RETURN stack
- Predictable performance, always two clock cycles per instruction, up to 200 MHz or 100 MIPS in a Virtex-II Pro FPGA
- Fast interrupt response; worst-case 5 clock cycles
- Optimized for Xilinx Spartan-3, Virtex-II, and Virtex-II Pro FPGA architectures just 96 slices and 0.5 to 1 block RAM
- Assembler, instruction-set simulator support

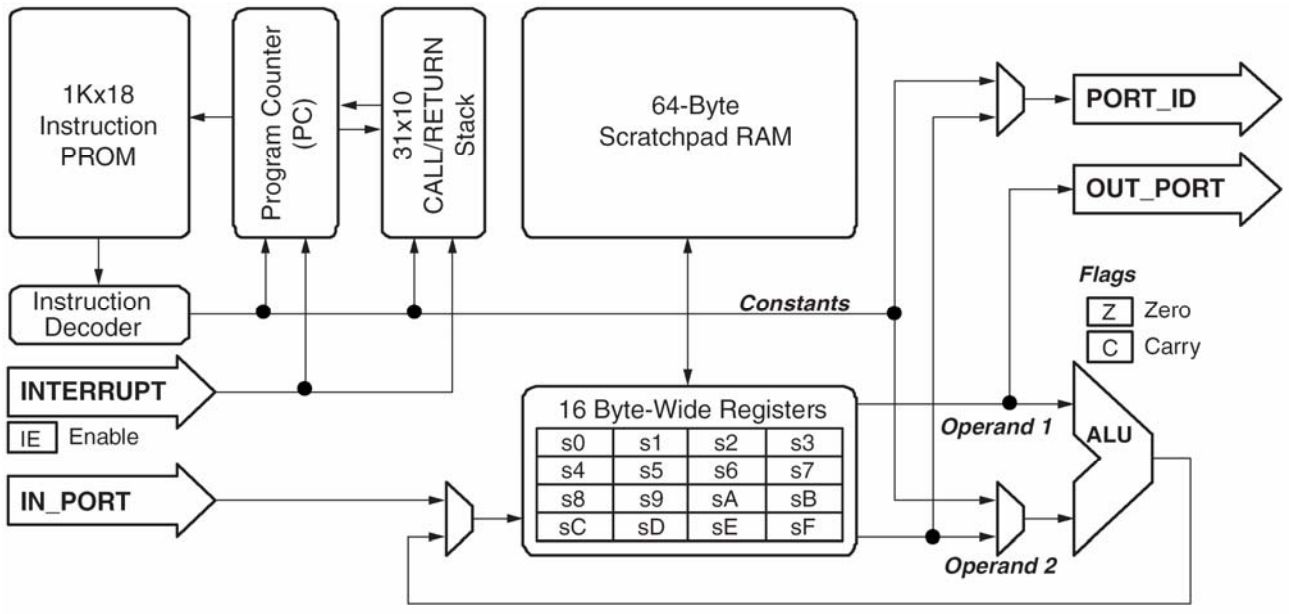


Figure 2.1 – PicoBlaze embedded microcontroller Block Diagram

2.3. Implementation of core and Parallel I/O with interrupts

The block diagram in figure 2.2 show the PicoBlaze connected with instruction ROM and the I/O interface for read and writes 8-bit's values (HEX 00 - FF) to the ports P0, P1, P3, P4 and P5 and write 16-bit's addressed (HEX 0000 - FFFF) out to port 2 compared with the purple I/O ports block and blue address decoder block at the BGEPB1 emulation 8051 peripherals block respectively in the diagram in figure 1.1 page 8. The system is created with three external interrupt pins at port 1 to receive external interrupts from hardware, example a keyboard switch or some other hardware inputs.

The PicoBlaze processor core communicate to the Input/Output Ports block (I/O block) via OUT_PORT which is an 8-bit data transmit out of the microcontroller in an internal pipeline to the I/O block. To receive data the internal pipeline IN_PORT is used which receive 8-bit data value from the I/O block. The PORT_ID is the port identity to chose the right channel for read or write via the internal pipeline, it is possible to control up to 256 I/O ports.

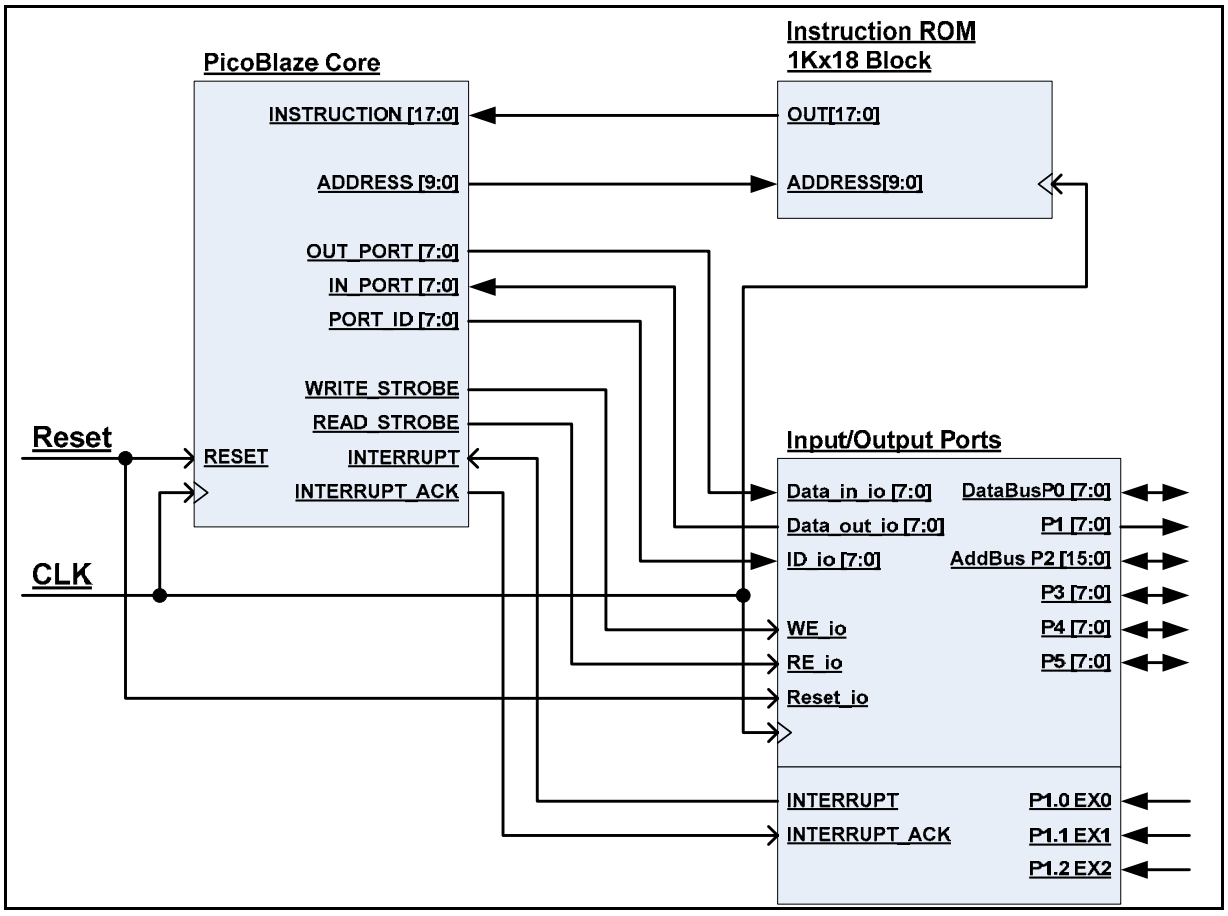


Figure 2.2 – Block diagram over I/O interface

2.5. VHDL code for the I/O Ports interface

The VHDL code is program in Xilinx project navigator, made from the BGEPB1 Special Function Register (SFR) shown in the requirement specifications in chapter 1.4 table 1.1 at page 9. The complete code is shown in appendix A page 50, under this text there are a cut-out of the reset routines from the I/O interface code.

Reset to standard value:

This code show the value of the I/O ports after reset where all ports will be set to high impedance level or synthesizable tri-state buffer. The Address bus is set to Hexadecimal 0000 this means the address bus will point on the memory at address 0, the maximum size of addressable memory will be 2^{16} = 65336 or 64Kbyte.

```
113   begin
114      process
115      begin
116
117         wait until (CLK_io'event and CLK_io='1');
118         X0_int <= Port_3_io(0); -- Read the Port 1 bit 0 value and save
119                                 -- it in the Internal signal.
120         X1_int <= Port_3_io(1); -- Read the Port 1 bit 1 value and save
121                                 -- it in the Internal signal.
122         X2_int <= Port_3_io(2); -- Read the Port 1 bit 2 value and save
123                                 -- it in the Internal signal.
124   ------------------------------------------------------------------------
125   --    RESET:
126   --    This function set ports level after reset and
127   --    define the value for variables
128   ------------------------------------------------------------------------
129         if(Reset_io='1') then
130            -- PORTS  level after reset
131            Port_0_io <= "ZZZZZZZZ";  -- Set Port 0 to high impedance level
132            Port_1_io <= "ZZZZZZZZ";  -- Set Port 1 to high impedance level
133            Port_2_io <= X"0000";     -- Set Port 2 to Address 0 (Hexadecimal)
134            Port_3_io <= "ZZZZZZZZ";  -- Set Port 3 to high impedance level
135            Port_4_io <= "ZZZZZZZZ";  -- Set Port 4 to high impedance level
136            Port_5_io <= "ZZZZZZZZ";  -- Set Port 5 to high impedance level
```

Figure 2.3 – VHDL code for reset data value for I/O interface to default.

Write and Read to I/O ports:

The I/O interface code looks at the Read- or Write-enable input and the ID input and uses Data-in and Data-out for transmit and receive data from this unit. As an example for transmit data to Port 0 also called DataBus, the programe uses the ID named (ID_io) HEX 01, the Write Enable named (WE_io) and the Data in named (Data_in_io) to write to Port 0. When the statement is true the processor will transmit the data value out on port 0. For receive data from Port 0 it is necessary to send a Read Enable named (RE_io) and the processor can read the value. The same in force for Port 1, Port 3, Port 4 and Port 5 but Port 2 is different because this is a 16-bit Address bus for sending data to this port it is necessary to send the data in two parts. First the low byte and second the high byte using the ID HEX 3 and HEX 04. This port can only transmit data and not receive anyone.

```
166   ----------------------------------------------------------------------------
167   --    Input & Output Interface:
168   --    This program function looks at the incoming port address from the
169   --    PicoBlaze processor core and control via write enable (WE_io) and
170   --    the read enable (RE_io) for Read or write to the Ports the ID is
171   --    control from the processor and set the SFR value to the right port.
172   ----------------------------------------------------------------------------
173   --     Port 0 I/O - SFR P0, HEX 01 DataBus
174         elsif(WE_io = '1'  and ID_io = X"01") then
175            Port_0_io <= Data_in_io; -- Send data from Microcontroller to Port 0
176         elsif(RE_io  = '1' and ID_io = X"01") then
177            Data_out_io <= Port_0_io;-- Send data from Port 0 to Microcontroller
178   --     Port 1 I/O - SFR P1, HEX 02
179         elsif(WE_io = '1'  and ID_io = X"02") then
180            Port_1_io <= Data_in_io; -- Send data from Microcontroller to Port 1
181         elsif(RE_io  = '1' and ID_io = X"02") then
182            Data_out_io <= Port_1_io;-- Send data from Port 1 to Microcontroller
183   --     Port 2 I/O - SFR P2, HEX 03 Address Bus Low and HEX 04 Address Bus High
184         elsif(WE_io = '1'  and ID_io = X"03") then
185            Port_2_io(7 DOWNTO 0) <= Data_in_io(7 DOWNTO 0); -- Send low byte to Port 2
186         elsif(WE_io = '1'  and ID_io = X"04") then
187            Port_2_io(15 DOWNTO 8) <= Data_in_io(7 DOWNTO 0);-- Send high byte to Port 2
188   --     Port 3 I/O - SFR P3, HEX 05
189         elsif(WE_io = '1'  and ID_io = X"05") then
190            Port_3_io <= Data_in_io; -- Send data from Microcontroller to Port 3
191         elsif(RE_io  = '1' and ID_io = X"05") then
192            Data_out_io <= Port_3_io;-- Send data from Port 3 to Microcontroller
193   --     Port 4 I/O - SFR P4, HEX 06
194         elsif(WE_io = '1'  and ID_io = X"06") then
195            Port_4_io <= Data_in_io; -- Send data from Microcontroller to Port 4
196         elsif(RE_io  = '1' and ID_io = X"06") then
197            Data_out_io <= Port_4_io;-- Send data from Port 4 to Microcontroller
198   --     Port 5 I/O - SFR P5, HEX 07
199         elsif(WE_io = '1'  and ID_io = X"07") then
200            Port_5_io <= Data_in_io; -- Send data from Microcontroller to Port 5
201         elsif(RE_io  = '1' and ID_io = X"07") then
202            Data_out_io <= Port_5_io;-- Send data from Port 5 to Microcontroller
```

Figure 2.4 – VHDL code for Transmit and Receive data to I/O interface.

Interrupt System:

The Interrupt system is used to control the external and internal interrupts build up after the principle from the 8051 microcontroller standard. The register is modified and there are used different Special Function Register (SFR) value compared with an 8051.

In figure 2.5 and 2.6 are the Interrupt Enable register IEN0 and IEN1 shown in this register it is possibility to activate and deactivate interrupts only the Watch Dog Timer (WDT) is not possible to disable after the enable. The Enable All (EA) enables all interrupts or disables all interrupts.

IEN0

EA	WDT	ET2	ET1	ET0	EX2	EX1	EX0

Figure 2.5 – Special Function Register IEN0

Bit	Function
EX0	Enables or disables external interrupt 0. If EX0 = 0, external interrupt 0 is disabled.
EX1	Enables or disables external interrupt 1. If EX1 = 0, external interrupt 1 is disabled.
EX2	Enables or disables external interrupt 2. If EX2 = 0, external interrupt 2 is disabled.
ET0	Enables or disables the timer 0 overflow interrupt. If ET0 = 0, the timer 0 interrupt is disabled.
ET1	Enables or disables the timer 1 overflow interrupt. If ET1 = 0, the timer 1 interrupt is disabled.
ET2	Enables or disables the timer 2 overflow interrupt. If ET2 = 0, the timer 2 interrupt is disabled. **(This bit is not used in this version)**
WDT	Enables the Watch Dog Timer overflow interrupt. If WDT = 1, the timer is activate and can not disables with out hardware reset.
EA	Enables or disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

Table 2.1 – List over Special Function Register IEN0

IEN1

						EC0	ES0

Figure 2.6 – Special Function Register IEN1

Bit	Function
ES0	Enables or disables Serial interrupt 0. If ES0 = 0, Serial interrupt 0 is disabled.
EC0	Enables or disables CAN-BUS interrupt 0. If EC0 = 0, CAN-BUS interrupt 0 is disabled.

Table 2.2 – List over Special Function Register IEN1

Note: The hatch last six bits is reserve for next version.

The Interrupt Service Control (ISC0) HEX 0E sets a flag in this register. If an interrupt is activate it will be controlled by hardware. The flag is read and cleared by software in the Interrupt Service Routine it is cleared bitwise in the ISC0 Special Function Register show in figure 2.7.

ISC0

IC0	IS0	IT2	IT1	IT0	IX2	IX1	IX0

Figure 2.7 – Special Function Register ISC0

Bit	Function
IX0	Read FLAG for external Interrupt 0 If IX0 = 1, external interrupt 0 is set. Clear the FLAG in Interrupt service routine with set bit IX0 to 0.
IX1	Read FLAG for external Interrupt 1 If IX1 = 1, external interrupt 1 is set. Clear the FLAG in Interrupt Service Routine with set bit IX1 to 0.
IX2	Read FLAG for external Interrupt 2 If IX2 = 1, external interrupt 2 is set. Clear the FLAG in Interrupt Service Routine with set bit IX2 to 0.
IT0	Read FLAG for Timer 0 overflow Interrupt. If IT0 = 1, Timer 0 overflow Interrupt is set. Clear the FLAG in Interrupt Service Routine with set bit IT0 to 0.
IT1	Read FLAG for Timer 1 overflow Interrupt. If IT1 = 1, Timer 1 overflow Interrupt is set. Clear the FLAG in Interrupt Service Routine with set bit IT1 to 0.
IT2	Read FLAG for Timer 2 overflow Interrupt. If IT2 = 1, Timer 2 overflow Interrupt is set. Clear the FLAG in Interrupt Service Routine with set bit IT2 to 0. **(This bit is not used in this version)**
IS0	Read FLAG for external Interrupt 2 If IX2 = 1, external interrupt 2 is set. Clear the FLAG in Interrupt Service Routine with set bit IX2 to 0.
IC0	Read FLAG for external Interrupt 2 If IX2 = 1, external interrupt 2 is set. Clear the FLAG in Interrupt Service Routine with set bit IX2 to 0.

Table 2.3 – List over Special Function Register ISC0

The VHDL code for the Interrupt System and handling of external interrupts is show in figure 2.8 under this text, to enable interrupts it is necessary at write to SFR HEX 0C named IEN0 where EA (Enable All) activates the MSB and the interrupts which are needed, example EX0 and the value to the register will be HEX 81 to set MSB and LSB. When there receives a interrupt on EX0 the function in line 318 set a flag in the Interrupt service control register the PicoBlaze read the flag and clear the afterwards.

```
273   ----------------------------------------------------------------------
274   --  INTERRUPT SYSTEM:
275   --  This program handle the Interrupt System there use the three register
276   --  named IEN0 - Interrupt Enable 0, IEN1 - Interrupt Enable 2 and ISC0 -
277   --  Interrupt Service Control. Activate with help of the SFR, the IEN0
278   --  and IEN1 enables the interrupt and the ISC0 show the status for the
279   --  interrupts.
280   ----------------------------------------------------------------------
281   --    INTERRUPT ENABLES - SFR IEN0, HEX 0C
282   --     ------------------------------------------
283   --    | EA  | WDT | ET2 | ET1 | ET0 | EX2 | EX1 | EX0 |
284   --     ------------------------------------------
285         elsif(WE_io  = '1' and ID_io = X"0C") then
286            EA_int <= Data_in_io(7);  -- Activate or deactivate all Interrupts EA
287   --       WDT_int <= Data_in_io(6);  -- Activate or deactivate WDT
288   --       ET2_int <= Data_in_io(5);  -- Activate or deactivate Interrupts Timer 2
289            ET1_int <= Data_in_io(4);  -- Activate or deactivate Interrupts Timer 1
290            ET0_int <= Data_in_io(3);  -- Activate or deactivate Interrupts Timer 0
291            EX2_int <= Data_in_io(2);  -- Activate or deactivate External Interrupt 2
292            EX1_int <= Data_in_io(1);  -- Activate or deactivate External Interrupt 1
293            EX0_int <= Data_in_io(0);  -- Activate or deactivate External Interrupt 0
294   --    INTERRUPT ENABLES - SFR IEN1, HEX 0D
295   --     ------------------------------------------
296   --    |  X  |  X  |  X  |  X  |  X  |  X  | EC0 | ES0 |
297   --     ------------------------------------------
298         elsif(WE_io  = '1' and ID_io = X"0D") then
299   --       EC0_int <= Data_in_io(1);  -- Activate or deactivate CAN-BUS Interrupt
300            ES0_int <= Data_in_io(0);  -- Activate or deactivate Serial Interrupt
301
302   --    INTERRUPT SERVICE CONTROL - SFR ISC0, HEX 0E
303   --     ------------------------------------------
304   --    | IC0 | IS0 | IT2 | IT1 | IT0 | IX2 | IX1 | IX0 |
305   --     ------------------------------------------
306         elsif(WE_io  = '1' and ID_io = X"0E") then
307            IC0_int <= Data_in_io(7);  -- Clear Interrupt FLAG for CAN-BUS
308            IS0_int <= Data_in_io(6);  -- Clear Interrupt FLAG for Serial
309            IT2_int <= Data_in_io(5);  -- Clear Interrupt FLAG for Timer 2
310            IT1_int <= Data_in_io(4);  -- Clear Interrupt FLAG for Timer 1
311            IT0_int <= Data_in_io(3);  -- Clear Interrupt FLAG for Timer 0
312            IX2_int <= Data_in_io(2);  -- Clear Interrupt FLAG for External 2
313            IX1_int <= Data_in_io(1);  -- Clear Interrupt FLAG for External 1
314            IX0_int <= Data_in_io(0);  -- Clear Interrupt FLAG for External 0
315         elsif(RE_io  = '1' and ID_io = X"0E") then
316            Data_out_io <= (IC0_int & IS0_int & IT2_int & IT1_int & IT0_int & IX2_int & IX
      1_int & IX0_int);
317   --    External Interrupt Service Routine
318         elsif (EX0_int='1' and IX0_int='0' and X0_int='1' and EA_int='1') then
319           IX0_int <= '1';        -- Set Interrupt FLAG
320           Interrupt_io <= '1';  -- Send Interrupt to PicoBlaze
321         elsif (EX1_int='1' and IX1_int='0' and X1_int='1' and EA_int='1') then
322           IX1_int <= '1';        -- Set Interrupt FLAG
323           Interrupt_io <= '1';  -- Send Interrupt to PicoBlaze
324         elsif (EX2_int='1' and IX2_int='0' and X2_int='1' and EA_int='1') then
325           IX2_int <= '1';        -- Set Interrupt FLAG
326           Interrupt_io <= '1';  -- Send Interrupt to PicoBlaze
```

Figure 2.8 – VHDL code for Interrupt System and external interrupts.

2.6. Test software in C code

This part describes and gives an example on a C code test program which is to test the communication between the C code language software and the Hardware description language. This is to test the ports for receiving and transmitting data and the interrupt request system. The C test program is written in Notepad and named prog_rom.c. The code is compiled by the PicoBlaze C Compiler PCCOMP alpha version 1.7.3 which is running in a DOS shell. The Program starts including the Spartan3.h file there is a part of the C compilers advanced function as read and write to I/O ports to use this function named OUTCHAR and INCHAR. The second file Included is the file named BGEPB1.h and this is the option file for the VHDL interface for the microcontroller (Special Function Register) for I/O ports and interrupts service shown in appendix B page 55.

IO_Ports - Notepad

File Edit Format View Help

```
//*****************************************************************************
//*COPYRIGHT: BENJAMIN GRYDEHOEJ - WWW.BG-ELEKTRONIK.DK - 2006 - TEST PROGRAM  *
//*****************************************************************************
//  Author:           Benjamin Grydehoej
//  Create the:       4th February, 2006
//  Last update the: 14th April, 2006
//  File:             prom_rom.c
//  Target Hardware: Xilinx Spartan3 - XC3S200
//  Tool chain:       Notepad - Microsoft Version 5.1
//  Compiler:         PCCOMP alpha 1.7.3 by Francesco Poderico
//  Version:          1.0.A
//
//  Test program for PicoBlaze (BGEPB1) MicroController.
//  The program test the I/O ports and the Interrupt register.
//*****************************************************************************
#include "lib\spartan3.h"       // Include PicoBlaze C compiler PCCOMP functions
#include "BGEPB1.h"             // Include BGEPS1 controller options (SFR)
// Add new variables for test program
unsigned char port_test1;
unsigned char port_test2;
unsigned char Interrupt_value;
void main(void)
{
  outchar(IEN0, 0x81);          // Set Enable All and external interrupt 0
  outchar(ISC0, 0x00);          // Clear the Interrupt Service Control Register
  port_test1 = 0;               // Set variable port_test 1 to zero
  port_test2 = 0;               // Set variable port_test 2 to zero
  while(1)
  {
   port_test2 = port_test2 + 1; // Add one to the variable port_test 2
   port_test1 = inchar(P3);     // Read HEX value on port 3 (slide switch 0 - 7 on Board) to  variable.
   outchar(P2L, 0x01);          // Send HEX value 01 to the low  byte of the AddressBus
   outchar(P2H, 0x02);          // Send HEX value 02 to the high byte of the AddressBus
   outchar(P0, port_test1);     // Write variable port_test1 value there are the input from port 0
   outchar(P1, 0xff);           // Write HEX value FF to Port 1
   outchar(P5, port_test2);     // Write variable port_test2 value to port 5
   Interrupt_value = inchar(ISC0); // Read the interrupt value from ISC0 register and save it in the
                                   // variable interrupt value.
   if(Interrupt_value == 0x01)  // If the Interrupt value is equal to HEX 01 (External interrupt 0)
    outchar(P4, port_test1);    // Write the slide switch value to Port 4 for test.
 }
}
```

Figure 2.9 – C code for Test of I/O ports and Interrupt System.

The Interrupt service control flag is just test with an IF statement which looks on the flag and send the value to port 5 if the flag is set. The reason that there is not used interrupt service routine in the test program is due to some problems with this function giving compiler errors when using the example from Francesco Poderico.

2.7. Simulation

The VHDL code for the BGEPB1 system and the C-code test program at page 20 is simulated in ModelSim with a view on Inputs and outputs which are accessed via the FPGA connections. Read more about this option for the test bench simulation in chapter 7.3 at page 45.

After reset of the FPGA the system will be initialized and will read and write to the ports as shown in figure 2.10, It is not possible to see the clock cycle in the simulation because one clock cycle is only 20nS and the simulation is shown from 0 to 25µS. after 4µS are the first data write out to the 16-bit address bus and afterwards the other ports will be written out after the structure in the code. Port 5 counts up, shown in the bottom of the simulation, and it will take only 6µS for each addition to the port.

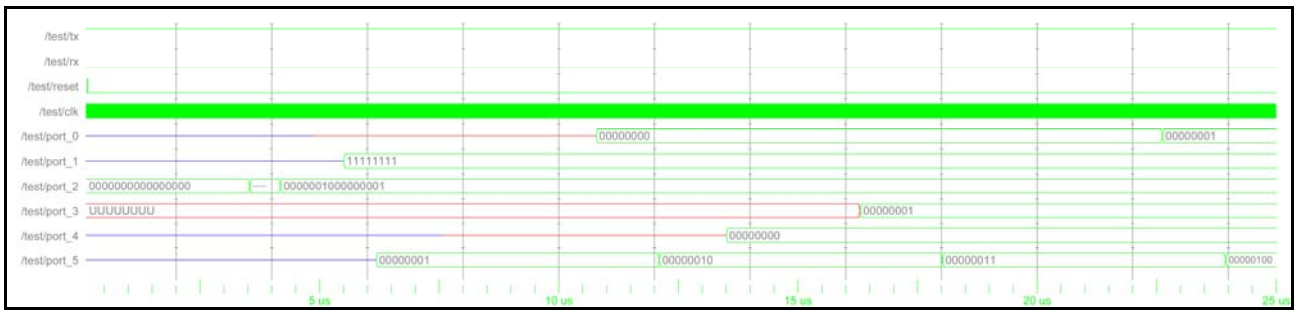
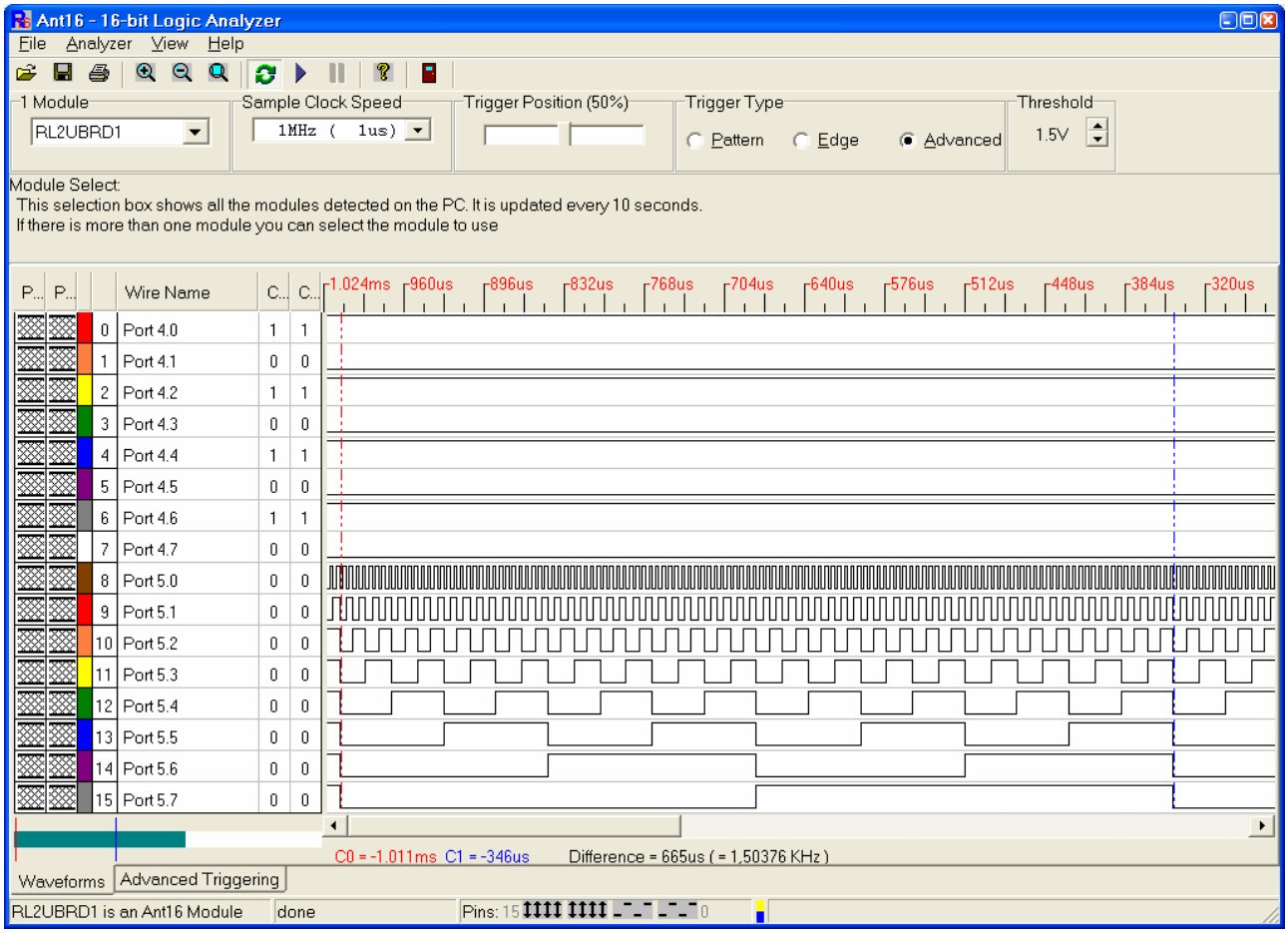


Figure 2.10 – Simulation of the program in ModelSim

2.8. Test and result

The practical test is done with a logic analyzer as in this example it is connected to port 4 and 5. Port 4 is the values send in from port 3 in this case the slide switch on the development board is set to HEX 55 and port 5 is run as a counter, counting up. Figure 2.11 shows the screenshot from the data analyzer program which has been used to check the output from the FPGA measurement on the development board. The port connection of the FPGA is shown in appendix C page 56.



Chapter 3:

Implementation of serial UART

3.1. Introduction

This chapter describes the option for the serial UART transmitter and Receiver Macros development by the company Ken Chapman, Xilinx Ltd. The macros package is created to run on following FPGA'S, Virtex, Virtex-E, Virtex-II, Spartan-II and Spartan-3. The macros provide the functionality of a simple UART transmitter and simple UART received each with the fixed characteristics of 1 start bit, 8 data bits (serially transmitted and received least significant bit first), No Parity and 1 stop bit.[3] This option makes it possible to communicate with a PC using a standard configuration the only thing needed setup for running the communication successfully is the Baud Rate timing which has been made adjustable in the SFR Register for the BGEPB1 microcontroller option.

Specification:

The standard baud rate the UART runs with is from 9600 and can support up to 115200. The serial UART operates after the standard with asynchronous receiver and transmitter that means the transmitter and receiver is not synchronised. The Serial UART contains an embedded 16 byte FIFO (First In First Out) buffer which just looks at the total size of data received or transmitted. The serial UART block diagrams for RX and TX is show in figure 3.1

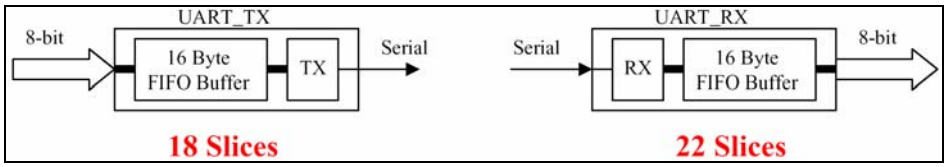
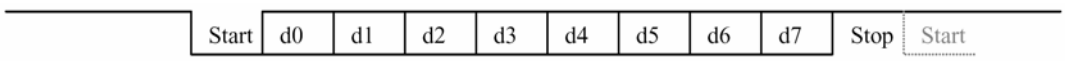


Figure 3.1 – Serial UART block diagrams.

The data is transmitted serially, LSB first, and given a bit rate from the BAUD rate. Since the transmitter can start sending this data at any time, the receiver needs a method of identifying when the first (LSB) is being sent. This is done with sending a Start bit as an active low start signal for the duration of one bit.



The receiver uses the falling edge from the Start bit to indicate that a new byte is ready to be received. After the last data bit MSB is received check to see if the transmitted stop bit is high as expected in the confirmation for the UART.

3.2. Implementation of serial UART

The block diagram in figure 3.2 show the implementation of the serial UART connected via the I/O interface block controlling the option of Special Function Register for serial data speed “BAUD rate” using the register named TLBS and THBS. The serial interrupt for receiving data plus the serial status register flag (SCON) which looks on the buffer status.

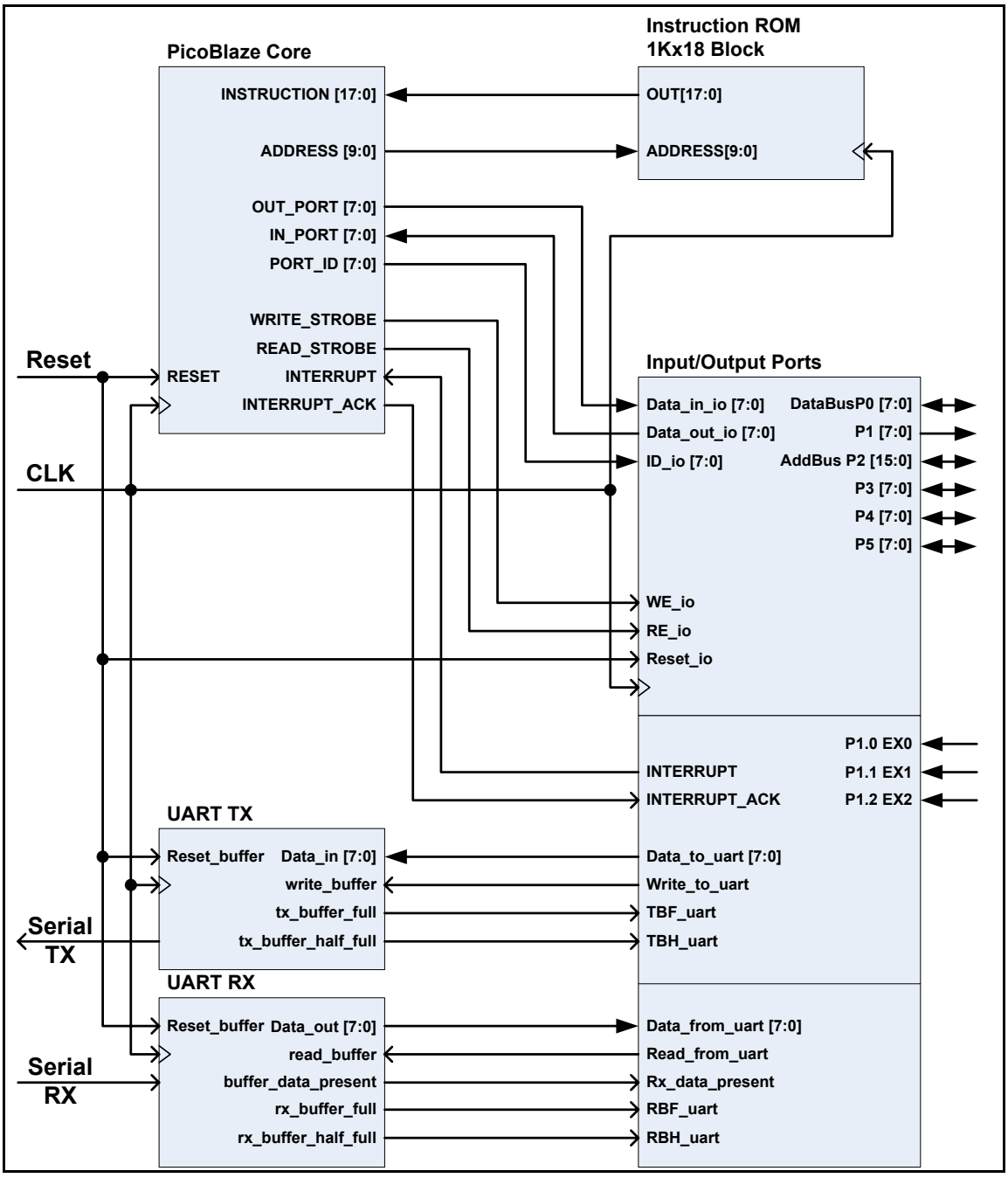


Figure 3.2 – Block diagram with TX and RX UART.

Read and write to UART:

The Serial UART communicates via the serial buffer for transmitting and receiving data named Serial Buffer “SBUF” and this is control by the SFR value HEX 08. For transmitting data via TX UART and receiving data via RX UART will the SFR value 08 be activated via the C-language program that writes or reads via SBUF. The VHDL code there interface this are showed in line 216 to 222 in figure 3.3.

```
207   ---------------------------------------------------------------------------
208   --  SERIAL DATA CONTROL:
209   --  This function controls the Serial UART with write and read via SBUF and
210   --  the option for the BAUD rate timing there control the speed for the serial
211   --  communication. The Serial Channel Control register SCON do it possible to
212   --  read the status flag for the receive and transmit buffer and look after
213   --  the status flag BDP for receive data.
214   ---------------------------------------------------------------------------
215   --    SBUF Read and write to Comport - SFR SBUF, HEX 08
216         elsif(WE_io = '1'  and ID_io = X"08") then -- Write to Serial Buffer
217            write_to_uart <= WE_io;         -- Enable write to UART
218            data_to_uart  <= Data_in_io;    -- Send Data to the UART buffer from
219                                            -- the SFR register named SBUF
220         elsif(RE_io  = '1' and ID_io = X"08") then -- Read to Serial Buffer
221            read_from_uart <= RE_io;        -- Enable read to UART
222            Data_out_io <= data_from_uart; -- Read Data from the UART buffer to
223                                            -- the SFR register named SBUF
224   --    Timer Baud rate serial, low byte - SFR TLBS, HEX 09
225         elsif(WE_io = '1'  and ID_io = X"09") then -- Write the low byte
226            TBS_uart(7 DOWNTO 0) <= Data_in_io; -- Send the low data byte to the Timer
227                                            -- Baud rate Serial for BAUD rate timing
228   --    Timer Baud rate serial, high byte - SFR THBS, HEX 0A
229         elsif(RE_io  = '1' and ID_io = X"0A") then -- Write the high byte
230            TBS_uart(15 DOWNTO 8) <= Data_in_io;-- Send the high data byte to the Timer
231                                            -- Baud rate Serial for BAUD rate timing
232
233   --    Serial Channel Control Register - SFR SCON, HEX 0B
234   --     -----------------------------------------------
235   --    |  X  |  X  |  X  | BDP | RBH | RBF | TBH | TBF |
236   --     -----------------------------------------------
237         elsif(RE_io  = '1' and ID_io = X"0B") then  -- Read the status flag from Serial
238                                                    -- Channel Control Register SCON
239            Data_out_io <= ("000" & BDP_uart & RBH_uart & RBF_uart & TBH_uart & TBF_uart);
240
241   -- Serial Interrupt handling
242         elsif (ES0_int='1' and IS0_int='0' and rx_data_present='1' and EA_int='1') then
243           IS0_int <= '1'; -- Set Serial Interrupt FLAG
244
```

Figure 3.3 – VHDL code for Serial interface of UART

BAUD Rate Timing:

The baud rate timer is a 16-bit timer that is controlled by a low and high byte send to the SFR register, via the Timer Low byte Baud rate Serial "TLBS" and Timer High byte Baud rate Serial "THBS". The baud rate is calculate out from the clock frequency on the FPGA board in this example the board is running with 50MHz and the baud rate is set to 9600Hz.

Calculation of value for BAUD Rate Timer:

$$Timer\_value = \frac{Clock\_frequency}{(16 \cdot BAUD\_RATE)} \Rightarrow \frac{50{,}000{,}000Hz}{(16 \cdot 9600Hz)} = 325.52 \approx \underline{\underline{326}}$$

The nearest integer is 326 this will in excess of the required tolerance equivalent baud rate of 9586Hz which is just 0.15%. Anything within 1% is really going to work as it allows for inaccurate clock rates and really poor switching in the serial lines. The HEX value for the baud rate timer will be HEX 0146 the low byte 0x46 and the high byte 0x01. This value is also the standard settings with reset of the system until there is reloaded a new value to the system via the special function register.

Calculation of the most common used baud rates with PC communication used on a FPGA with a clock frequency at 50MHz:

BAUD Rate:	Result:	Value in Integer:	Value in HEX:	Tolerance:
9600	325.52	326	0146	0.147%
19200	162.76	163	00A3	0.147%
38400	81.38	81	0051	0.469%
57600	54.25	54	0036	0.469%
115200	27.12	27	001B	0.469%

Table 3.1 – List over standard BAUD Rate used in a FPGA there run with a frequency at 50MHz.

The HEX value for the baud rate timer is loaded via SFR value HEX 09 for the Low byte and 0A for the high byte. The value is loaded to the TBS_uart variable in the VHDL code show in figure 3.3 from line 224 to line 230 at Page 25. The baud rate timer counter code in VHDL is showed in figure 3.4 at page 27.

```
289   ---------------------------------------------------------------------------
290   -- BAUD RAGE TIMER:
291   -- This function sets the baud rate to the value for the UART communications,
292   -- after reset is the value default integer 326 or HEX value 0146 a speed at
293   -- 9600 baud (bit per sec.)
294   ---------------------------------------------------------------------------
295     baud_timer: process(clk)
296     begin
297       if clk'event and clk='1' then  -- Wait for a clock event
298         if baud_count = TBS_uart then-- Compare the baud count with the TBS uart
299             baud_count <= 0;         -- If this is true will the baud count be set
300           en_16_x_baud <= '1';       -- to zero and enable en_16_x_baud with a one.
301          else
302             baud_count <= baud_count + 1; -- If this not are true the counter will
303           en_16_x_baud <= '0';            -- add one and the en_16_x_baud will be
304         end if;                           -- disable.
305       end if;
306     end process baud_timer;
```

Figure 3.4 – VHDL code for Serial UART timer

Serial Status Register:

The serial status register sets flag for the TX/RT buffer and for the BDP flag for new receive data in the RX buffer. The explanation of the flag function is shoved in table 3.2.

SCON

X	X	X	BDP	RBH	RBF	TBH	TBF

Figure 3.5 – Special Function Register SCON.

Bit	Function
TBF	Read FLAG Transmission Buffer Full there is set by hardware in TX UART. If TBF = 1, Flag Transmission Buffer Full is set. When the 16-byte FIFO buffer is full, this output becomes active HIGH. The host system should not attempt to write any new data until the serial transmission has been able to create a space. Any attempt to write data will mean that the new data is ignored.
TBH	Read FLAG Transmission Buffer Half full there is set by hardware in TX UART. If TBH = 1, Flag Transmission Buffer Half full is set. When the 16-byte FIFO buffer holds eight or more bytes of data waiting to be transmitted, this output becomes active HIGH. This is a useful indication to the host system that the FIFO buffer is approaching a full condition, and that it would be wise to reduce the rate at which new data is being written to the macro.
RBF	Read FLAG Receiving Buffer Full there is set by hardware in RX UART. If RBF = 1, Flag Receiving Buffer Full is set. When the 16-byte FIFO buffer is full, this output becomes active HIGH. The host system should rapidly respond to this condition by reading some data from the buffer so that further serial data is not lost.
RBH	Read FLAG Receiving Buffer Half full there is set by hardware in TX UART. If RBH = 1, Flag Receiving Buffer Half full is set. When the 16-byte FIFO buffer holds eight or more bytes of data waiting to be read, this output becomes active HIGH. This is a useful indication to the host system that the FIFO buffer is approaching a full condition, and that it would be wise to read some data in the very near future.
BDP	Read FLAG for Receiving Buffer Data Present. If BDP = 1, Receiving Buffer Data Present is set. When the internal buffer contains one or more bytes of received data this signal will become active HIGH and valid data will be available to read

Table 3.2 – List over Special Function Register SCON

3.3. Simulation

The simulation is made out form the test program in figure 3.6 which sends serial data out onto the comport with a baud rate at 115200 after having received a ASCI value ‘s’ from e.g. a computer using the HyperTerminal. The value ‘s’ starts the transmission and the program sends the value “HEY” to the computer via serial communication. With help from P4 and P5 it is possible to watch the Interrupt status and the buffer status. The program is tested in ModelSim but it is not easy to show on paper because the transmission occurs over a lot of clock cycles and will not give much sense.

prog_rom - Notepad

File Edit Format View Help

```
//*****************************************************************************
//*COPYRIGHT: BENJAMIN GRYDEHOEJ - WWW.BG-ELEKTRONIK.DK - 2006 - TEST PROGRAM  *
//*****************************************************************************
//  Author:           Benjamin Grydehoej
//  Create the:       4th February, 2006
//  Last update the: 14th April, 2006
//  File:             prom_rom.c
//  Target Hardware: Xilinx Spartan3 - XC3S200
//  Tool chain:       Notepad - Microsoft Version 5.1
//  Compiler:         PCCOMP alpha 1.7.3 by Francesco Poderico
//  Version:          1.0.A
//
//  Test program for PicoBlaze (BGEPB1) MicroController.
//  The program test the Serial port with receive and transmit.
//*****************************************************************************
#include "lib\spartan3.h"       // Include PicoBlaze C compiler PCCOMP functions
#include "BGEPB1.h"             // Include BGEPS1 controller options (SFR)
// Add new variables for test program
unsigned char serial_value;
unsigned char Interrupt_status;
unsigned char buffer_status;
void main(void)
{                       // Set baud rate to 115200Hz
  outchar(THBS, 0x00);  // Set High byte for timer baud rate
  outchar(TLBS, 0x1B);  // Set Low byte for timer baud rate
  outchar(IEN0, 0x80);  // set Enable All Interrupt
  outchar(IEN1, 0x01);  // Activate Serial Interrupt
  while(1)
  {
   serial_value = inchar(SBUF);// Read value from Serial buffer
   if(serial_value == 0x73)     // If serial value equal to 's' (start)
   {
    outchar(SBUF, 'H');         // Print "HEY" to the serial port
    outchar(SBUF, 'E');
    outchar(SBUF, 'Y');
    Interrupt_status = inchar(ISC0);    // Read Interrupt flag
    buffer_status    = inchar(SCON);    // Read Buffer Status
    outchar(P4, Interrupt_status);      // Write Interrupt Flag to P4
    outchar(P5, buffer_status);         // write buffer status to P5
   }
  }
}
```

Figure 3.6 – Test program for serial UART

3.4. Test and result

In practical the HyperTerminal is used, as shown in figure 3.7, to transmit and receive the test data. For watching the Interrupt flag and the buffer status the data analyzer is connected to port 4 and port 5. The Serial UART is tested with success.

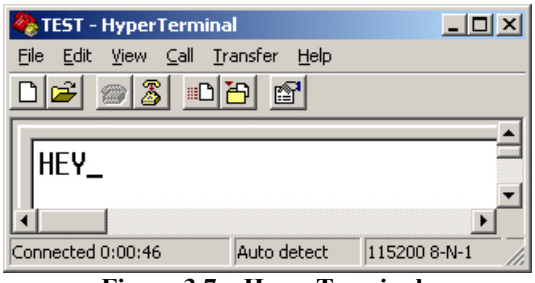


Figure 3.7 – HyperTerminal

Chapter 4:

Implementation of Timers

4.1. Introduction

This chapter describes the two implemented timers in the BGEPB1 the Timer 0 which is an 8-bit timer and Timer 1 which is a 16-bit timer. Both timers work as the count up with the clock frequency, the timer interval depend on the clock frequency. It is possible to start and stop the timer from the SFR named TCON and read the timer status in the same register with look on timer flag to check the timer is running. The interrupt service routine can be active with the register IEN0 shown in chapter 2, I/O ports interface at page 17 with use of the interrupt enable ET0 and ET1. The register ISC0 from same chapter page 18 uses Interrupt Service Control to clear the Interrupt in the C code program and when the timer is equal to the set timer value it will activate the Interrupt and the C code program will be enable to response to the interrupt and after end reading the C code program will be enable to clear the interrupt and continue. The timer value is set by a timer register and will be reloaded every time there sends a new value to this register.

4.2. Implementation of Timers

The Timers is implemented in the block named Input/Output ports in the block diagram shown in chapter 3 at page 14 in figure 3.2 for communicate with the SFR to set timer value, start/stop timer and the Interrupt service control register. After system reset on the FPGA the timers will default be set to maximum value this will say the 8-bit timer is set to HEX FF or integer 255 and the 16-bit timer is set to HEX FFFF or integer 65535. Both timer will be stop after reset and shall starts via the TCON register.

The VHDL code in figure 4.1 show the option of the timer service control register made after the same method in chapter 2 for the interrupt control. Where the looks on the incoming register value and do option out from that.

```
299   ---------------------------------------------------------------------------
300   -- This program function support the Timer option via the SFR TCON there
301   -- start/stop timer to run and the reading flag function for timer interrupt.
302   -- The supports also the reload value for Timer 0 and Timer 2.
303   ---------------------------------------------------------------------------
304   --    TIMER SERVICE CONTROL - SFR TCON, HEX 0F
305   --     -----------------------------------------------
306   --    |  X  |  X  | TF2 | TR2 | TF1 | TR1 | TF0 | TR0 |
307   --     -----------------------------------------------
308         elsif(WE_io  = '1' and ID_io = X"0F") then  -- Write to TCON register
309                                                    -- for Start/Stop
310            TR0_timer <= Data_in_io(0);             -- Start timer 0 with set a '1'
311            TR1_timer <= Data_in_io(2);             -- Start timer 1 with set a '1'
312         elsif(RE_io  = '1' and ID_io = X"0F") then  -- Read status flag and which timer
313                                                    -- there are on!
314            Data_out_io <= ("0000" &  TF1_timer & TR1_timer & TF0_timer & TR0_timer);
315
316   --    Timer Count 0 - TC0, HEX 10
317         elsif(WE_io  = '1' and ID_io = X"10") then  -- Reload new value to timer 0
318            TC0_timer <= Data_in_io;
319   --    Timer Count 1 - TCL1, HEX 11 and TCH1 HEX 12
320         elsif(WE_io  = '1' and ID_io = X"11") then  -- Reload new value to Timer 2
321            TC1_timer(7 DOWNTO 0) <= Data_in_io;     -- Set the low byte
322         elsif(WE_io  = '1' and ID_io = X"12") then
323            TC1_timer(15 DOWNTO 8) <= Data_in_io;    -- Set the high byte
324
325   --    Timer Interrupt Service Routine
326         elsif (ET0_int='1' and TF0_timer='1' and IT0_int='0' and EA_int='1') then
327           IT0_int <= '1';       -- Set Timer 0 Interrupt FLAG
328           Interrupt_io <= '1'; -- Send Interrupt to PicoBlaze
329         elsif (ET1_int='1' and TF1_timer='1' and IT1_int='0' and EA_int='1') then
330           IT1_int <= '1';       -- Set Timer 1 Interrupt FLAG
331           Interrupt_io <= '1'; -- Send Interrupt to PicoBlaze
```

Figure 4.1 – VHDL code for Timer control.

The VHDL code in figure 4.2 shows the code for the counter used for timer 0. The code for timer 1 is exactly the same code the only difference between them is the use other variable names and the variable TC0 there is a 8 bit value for timer 0 and the variable TC1 is a 16 bit value for Timer 1.

```
307   ---------------------------------------------------------------------------
308   --  TIMER COUNTS SYSTEM:
309   --  This system controls the timer and set the timer flag when the timer is
310   --  run out.
311   ---------------------------------------------------------------------------
312     Timer0_control: process(clk)
313     begin
314       if(clk'event and clk='1' and TR0='1') then -- Wait for a clock event and
315         if Timer0 = TC0 then                     -- TR0 is set active to run
316            Timer0 <= 0;              -- Compare the Timer 0 with the TC0 value
317            TF0 <= '1';               -- If this is true will the timer count be set
318         else                         -- to zero and enable the Timer flag with a one.
319            Timer0 <= Timer0 + 1;     -- If this not are true the counter will
320            TF0 <= '0';               -- add one and the Timer flag will be
321         end if;                      -- disable.
322       end if;
323     end process Timer0_control;
```

Figure 4.2 – VHDL code for the Timer 0 Counter

Timer Register:

The timer register is built for controlling the Timer 0, Timer 1 and Timer 2. The timer 2 is not activated in version BGEPB1 but there is reserved space in the register for timer 2 to later version update. The register TR0, TR1 and TR2 Is for start timer to run with an active one and the TF0, TF1 and TF2 is timer flag which will be set when the timer is running out, this is equal to the set timer value and uses the flag for activate the interrupt too. In figure 4.3 is the Timer Service Control Register TCON shown.

TCON

X	X	TF2	TR2	TF1	TR1	TF0	TR0

Figure 4.3 – Timer Service Control Register TCON.

Bit	Function
TR0	Enable Timer Run 0 to start counting If TR0 = 1, The Timer 0 will Rune.
TF0	Read FLAG for Timer 0 If TF0 = 1, The Timer 0 is just count out. The FLAG will be set and clear by Hardware.
TR1	Enable Timer Run 1 to start counting If TR1 = 1, The Timer 1 will Rune.
TF1	Read FLAG for Timer 1 If TF1 = 1, The Timer 1 is just count out. The FLAG will be set and clear by Hardware.
TR2	Enable Timer Run 2 to start counting If TR2 = 1, The Timer 2 will Rune. **(This bit is not used in this version)**
TF2	Read FLAG for Timer 2 If TF2 = 1, The Timer 2 is just count out. The FLAG will be set and clear by Hardware. **(This bit is not used in this version)**

Table 4.1 – List over Special Function Register TCON.

Calculation of timer value:

The timing depends on the clock frequency and in this case the FPGA runs with 50MHz and the maximum timer value created for the Timer 0 and Timer 1 is calculate to 5.1µS for Timer 0 and 1.3107mS for Timer1 showed in the equation under this text.

$$Timer0\_value\_in\_Sec. = \frac{1}{\left(\frac{Clock\_frequency}{(Timer0\_value\_in\_dec)}\right)} \Rightarrow \frac{1}{\left(\frac{50{,}000{,}000Hz}{255}\right)} = \underline{\underline{5.1\mu S}}$$

$$Timer1\_value\_in\_Sec. = \frac{1}{\left(\frac{Clock\_frequency}{(Timer1\_value\_in\_dec)}\right)} \Rightarrow \frac{1}{\left(\frac{50{,}000{,}000Hz}{65535}\right)} = \underline{\underline{1.3107mS}}$$

This is an example to make a calculation of the timer value there is to be uploaded to the timer out from the expected time at 500µS. The value will be 25000 as shown in the equation under this text and therefore it is necessary to use the timer 1, a 16-bit timer, for this operation because of the high number.

$$Timer1\_value\_in\_dec = Clock\_frequency \cdot Timer1\_value\_in\_Sec. \Rightarrow 50{,}000{,}000 \cdot 500\mu S = \underline{\underline{25000}}$$

4.3. Simulation

A simulation is made on timer 0 and timer 1 out from the C program in figure 4.4. The program set timer 0 with the loaded HEX value 19, which is 500nS, and timer 1 is loaded to HEX value 32, being 1µS. Afterwards the timers are started and the timer flag and interrupt flag is shown on port 4 and 5. The timer will run until the program is stopped.

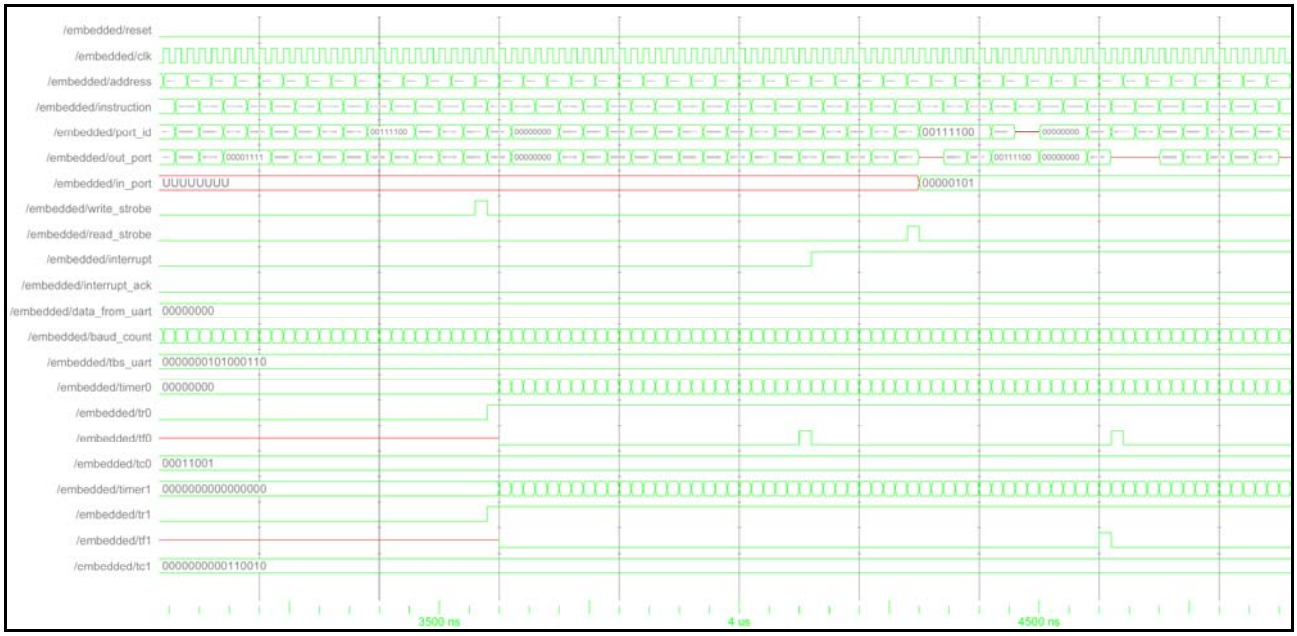
prog_rom - Notepad

File Edit Format View Help

```
//*************************************************************************************
//*COPYRIGHT: BENJAMIN GRYDEHOEJ - WWW.BG-ELEKTRONIK.DK - 2006 - TEST PROGRAM  *
//*************************************************************************************
//   Author:           Benjamin Grydehoej
//   Create the:       4th February, 2006
//   Last update the: 14th April, 2006
//   File:             prom_rom.c
//   Target Hardware: Xilinx Spartan3 - XC3S200
//   Tool chain:       Notepad - Microsoft Version 5.1
//   Compiler:         PCCOMP alpha 1.7.3 by Francesco Poderico
//   Version:          1.0.A
//
//   Test program for PicoBlaze (BGEPB1) MicroController.
//   The program test the Timer 0 and Timer 1.
//*************************************************************************************
#include "lib\spartan3.h"       // Include PicoBlaze C compiler PCCOMP functions
#include "BGEPB1.h"             // Include BGEPS1 controller options (SFR)
// Add new variables for test program
unsigned char Timer_Flag;
unsigned char Interrupt_status;
void main(void)
{
  outchar(TC0,  0x19);  // Set Count value for Timer 0 (500nS)
  outchar(TCL1, 0x32);  // Set Count value Low for Timer 1 (1uS)
  outchar(TCH1, 0x00);  // Set Count value High for Timer 1
  outchar(IEN0, 0x88);  // set Enable All Interrupt and Timer 0
  outchar(TCON, 0x05);  // Start Timer 0 and Timer 1
  while(1)
  {
   Timer_Flag       = inchar(TCON);  // Read value from Timer Flag
   Interrupt_status = inchar(ISC0);  // Read value from Interrupt Flag
   outchar(P4, Timer_Flag);          // Write Timer Flag to P4
   outchar(P5, Interrupt_status);    // write Interrupt status to P5
  }
 }
}
```

Figure 4.4 – Test program for timer 0 and 1.

The simulation shows the timing. Timer 0 sets a flag after 25 clock cycles which has a duration of 500nS, this is shown as TF0. Timer 1 set the first flag after 50 clock cycles on the timer 1 count shown in TF1. The timers raises one clock cycle every time the flag is set because there goes one clock cycle to clear the counter again. This is not useful and it is necessary to change this in the VHDL code or take care of that in the C program. The Interrupt goes high after the first Timer flag as is should, but because there are problems with the Interrupt Service Routine in the C compiler. It is not possible to auto clear the interrupt as expected with the Interrupt acknowledge. If it worked as expected there should be calculates with a response up to 6 clock cycle before the flag would be clear in worst case and these 6 clock cycles are also necessary to be taken care of in the C- or ASM-code timer programming.



4.4. Test and result

The program is tested on hardware by downloading the code to the development board and with help of the data analyzer it is possible to watch the timer flag and the status for the Interrupt Service Routine on port 4 and 5. But the timing is not exactly what is shown in the simulation because it takes a few extra clock cycles to write out on Port 4 and 5. But it gives an idea of how it should work correctly.

Chapter 5:

Implementation of Serial Flash ROM interface

5.1. About serial interface

The serial Flash PROM interface can be accessed through serial data communication from the FPGA via three data connections. Serial Data from Flash, Enable Serial Flash from FPGA and Clock signal from the FPGA. The Flash PROM can only be used as Program ROM or for fixed data as Ethernet MAC ID, ASCII data for display, encryption codes etc. All types are fixed values which are programmed into the flash via JTAG standard communication using the iMPACT tool from Xilinx Project Navigator, which is programmed with the file formats named Object (.mcs) or HEX (.hex). The JATG is a serial bus made for in-circuit test and programming using the four communications lines named Test Clock (TCK), Test Mode Select (TMS), Test Data In (TDI) and Test Data Out (TDO) connected to a external programming unit at the connector shown in the left side of figure 5.1.

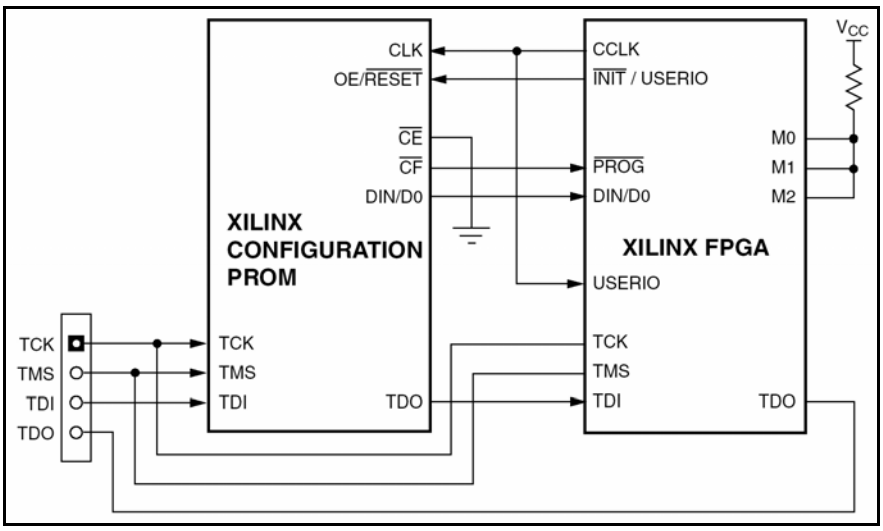


Figure 5.1 – Serial hardware interface

The VHDL code for accessing the PROM information via the FPGA is available from Xilinx's homepage as free code. The Serial Flash PROM is not used in this program because there is sufficient PROM for the code in the FPGA.

- XAPP694 Reading User Data from Configuration PROMs
  http://www.xilinx.com/xlnx/xweb/xil_publications_display.jsp?sGlobalNavPick=&sSecondaryNavPick=&category=-1209899&iLanguageID=1 or from the library Serial_Flash on the CD-ROM

Chapter 6:

Design of CAN-BUS Interface

6.1. Introduction

The Control Area Network (CAN) Bus interface is a serial asynchronous transmission scheme that uses a communication protocol which efficiently supports distribution of real time control with a very high level of security. The specification is defined with the ISO 11898 “OSI Model”. The CAN 2.0A is an extended message format defined in CAN 1.2 and CAN 2.0B describing both standard and extended message formats.

The Layer structure of the CAN BUS is compared with the seven OSI model layer showed in figure 6.1. The OSI layer is compress to four main layers for the CAN because some of the layer overlaps each other, the four CAN layer is Physical Layer, Transfer Layer, Object Layer and Application Layer these layer is describe in the four subjects under this text.

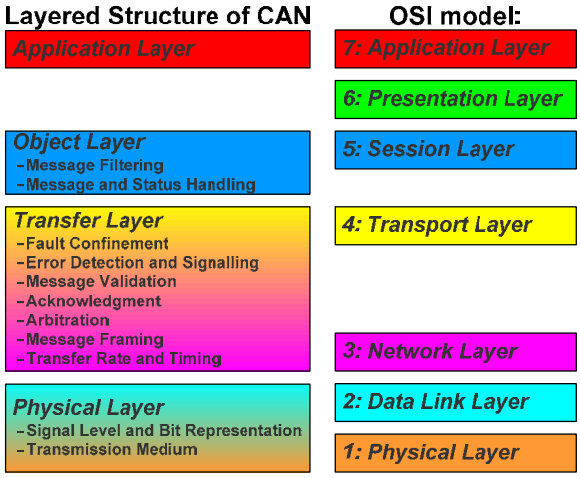


Figure 6.1 – CAN layer & OSI layer.

Physical Layer:

The physical layer is the hardware specifications for the CAN standard and use connector type as standard male 9-PINs SUB-D connector and the cable is typical Shielded Twisted Pair (STP) or Un-shielded (UTP) cables the characteristic for the line impedance is 120 Ohm, common mode voltage ranges from -2 Volts on CAN_L to +7 Volts on CAN_H. The balanced differential 2-wire CAN bus can transmitted signal up to 40 meters with a speed of 1Mbps and less at 1km up to 20Kbps. The CAN standard bit encoding use the system called Non Return to Zero (NRZ). The CAN transmits data through a binary model of dominant bits and recessive bits where dominant is logic 0 and recessive is logic 1. The maximum bits there most been send subsequent is five dominant or recessive, if more there will be set an extra bit there is reversed from the other bits.

Transfer Layer:

The Transfer layer handles the protocol for transmitting and receiving data via message transfer is manifested and controlled by four different frame types, the specification for CAN protocol 2.0A is shown in figure 6.1 and stated in bullets point.

- Start of frame
- The Arbitration field identifier the ID.
- The Control field consists of four bits Data length Code that identify how many Bytes there are in the data packet
- The Data field consists of the data to be transferred
- The Cyclic Redundancy Check (CRC) sequence is calculate from the Start Of Frame (SOF) field to and with the Data field, with the polynomial X15+X14+X10+X8+X7+X4+X3+1
- The ACK field acknowledgment a valid message received correctly
- End of frame

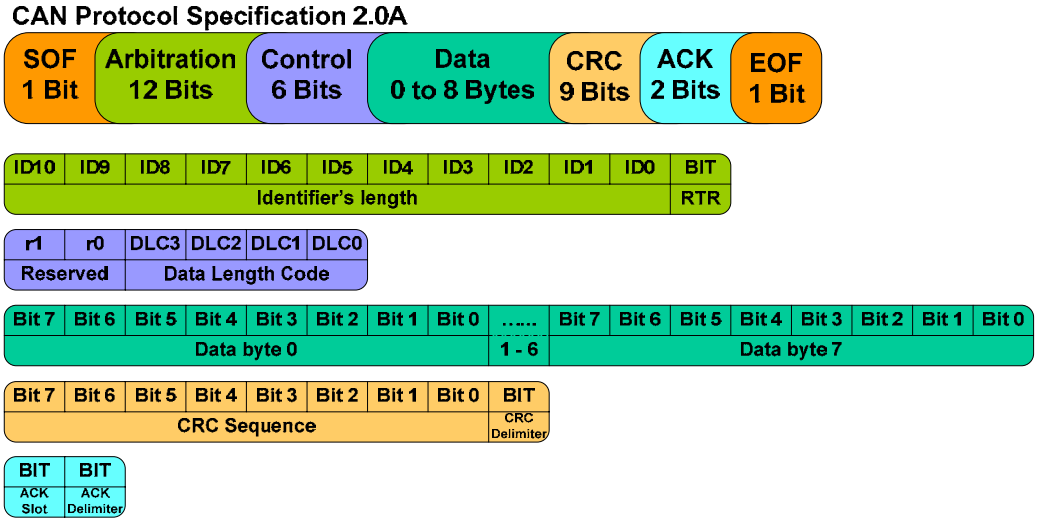


Figure 6.2 – CAN protocol Specification 2.0A

Object Layer:

The object layer handles the message filtering and the messages, the message filtering checks that the data packets are valid; there is a different between this function for either the transmitter or the receivers of the messenger. The status handling 5 different error types named Bit Error, Stuff Error, CRC Error, Form Error and Acknowledgment Error.

Application Layer:

The application layer handle the communication to the program code read and write to register, in this project are the SFR used.

6.2. Design of CAN-BUS Hardware Interface

The Hardware interface is build up on a Printed Circuit Board (PCB) and made out from the block diagram in figure 6.3. The interface board will be connected to the development board via an IDC header being the standard connector on the development board. The Interface board will be supplied with 3.3 volt power from the development board and there will be transmitted and received data via this header. The connection out to the world is a male 9-PINs SUB-D connector there is mounted with UTP cable.

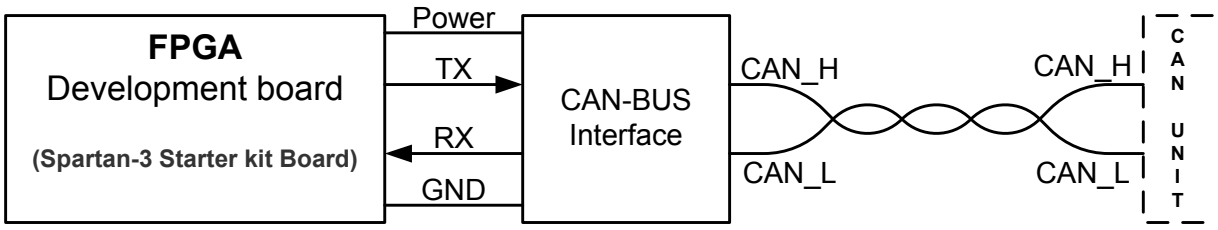


Figure 6.3 – Block diagram over CAN-BUS Interface.

Interface:

The design use a MAX3053 for interfaces between the CAN protocol from the FPGA and the physical wires of the bus lines in a CAN. The MAX3053 has three different modes of operation high-speed, slope control, and shutdown. High-speed mode allows data rates up to 2Mbps. In slope control mode, data rates are between 40kbps and 500kbps so the effects of EMI are reduced and unshielded twisted or parallel cable may be used. In shutdown mode, the transmitter is switched off, and the receiver is switched to a low-current mode.[4]

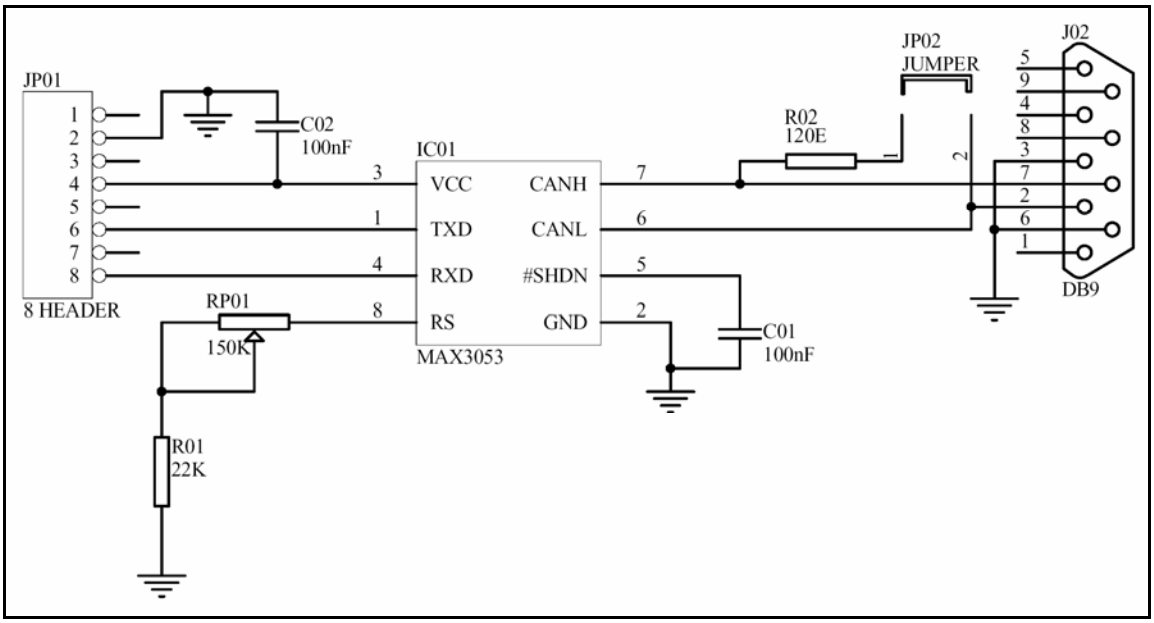


Figure 6.4 – Interface Circuit Diagram

Peripherals components:

The circuit in figure 6.4 at page 38 is made out from an application note in the data sheet page 1 for the device named MAX3053[4] the peripherals components is the C02 there is a ceramic decouple capacitor removing noise from the power supply lines.

Potentiometer PR01 and resister R01 is place to adjust the value from 22KΩ to 172KΩ the reason for making this adjustment is to get the line drivers to switch on and off as quickly as possible optimizing the limit of rise and fall slope of the data signal. Example with a speed at 500Kbps the resistor value will be 24KΩ shown in the data sheet page 4[4].

The capacitor C01 is mounted for hold the shutdown input pin high impended and the device will always be turned on to run. If the shutdown pin is set to low the device will go in the shutdown mode. The last features in the circuit is the jumper JP01 and the impedance resistor R02 at 120Ω, it the jumper is set the circuit will make an impedance termination for the CAN bus.

Design:

The circuit is made on a single side PCB using Surface-Mount Devices (SMD) and designed in Protel Design Explore 99 SE there are a full functional 30 days trial version of a professional PCB layout tool. The layout result is showed in figure 6.5 for the bottom layer to the left, the top over layer in the middle and the bottom over layer to the right.

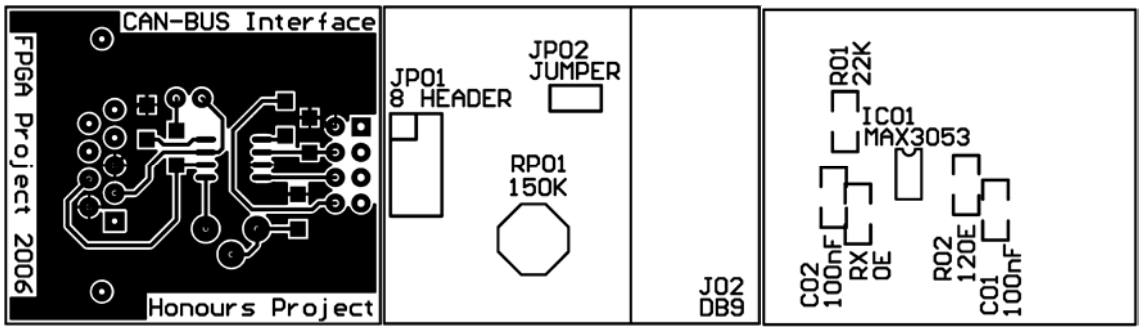


Figure 6.5 – Bottom layer, top over layer and bottom over layer.

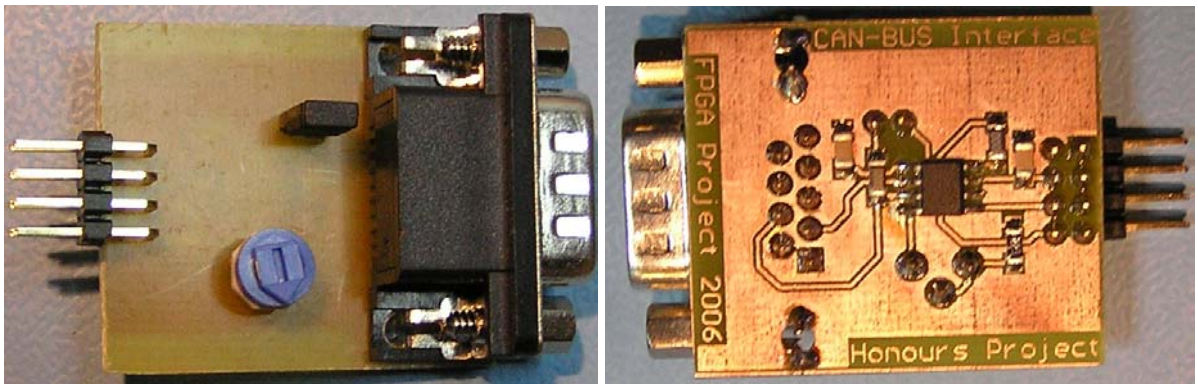


Figure 6.6 – Pictures of the CAN bus interface.

6.3. Design of CAN-BUS VHDL interface for transmitting

The CAN transmitter interface is designed as the VHDL part which has not been implemented in the BGEPB1 core at the moment because there is still missing some development. But the corner stones have been built to be able to send data test packets from the VHDL interface. The data packet is generated from the protocol; the ID address set to HEX 200 and four data byte set to HEX AA, FF, 00 and 55. The CRC calculation is done manually and gives the HEX value 69, this is all fixt value for the data packet. In this VHDL code the serial sequence is automatically generates as shown in figure 6.7.

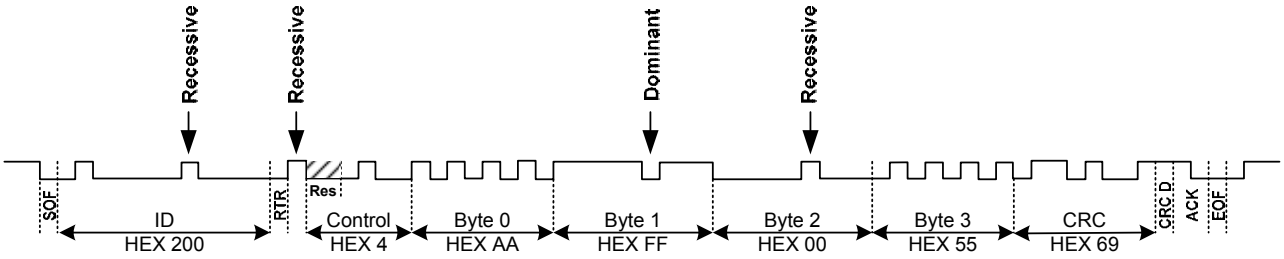


Figure 6.7 – Test Data packet for transmission via CAN.

The data value will be added together in a vector chosen as the worst case value of 95 bit according to the CAN specification 2.0A. When the data is “received”, in this code example the values are set to fixed values, the data would be added together chosen out after the value of bytes as shown in code line 138 to 155 in figure 6.8 at page 41. The unused bit in the vector is set to high and will be sending as high output to the CAN interface. The loop from line 160 to 183 in the VHDL code is a loop that inserts the recessive and the dominant bit after every five identically bit. The function in line 188 send data serial out to the CAN bus interface named TX_CAN at the output on FPGA.

```
137         -- Chose the packet format between 0 to 8 byte data transmitting
138         IF (Data_Length = 0) THEN
139           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & CRC
      & CRC_del & ACK_slot & ACK_del & "0" );
140         ELSIF (Data_Length = 1) THEN
141           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & CRC & CRC_del & ACK_slot & ACK_del & "0" );
142         ELSIF (Data_Length = 2) THEN
143           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & CRC & CRC_del & ACK_slot & ACK_del & "0" );
144         ELSIF (Data_Length = 3) THEN
145           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & DATA2 & CRC & CRC_del & ACK_slot & ACK_del & "0" );
146         ELSIF (Data_Length = 4) THEN
147           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & DATA2 & CRC & CRC_del & ACK_slot & ACK_del & "0" );
148         ELSIF (Data_Length = 5) THEN
149           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & DATA2 & DATA3 & DATA4 & CRC & CRC_del & ACK_slot & ACK_del & "0" );
150         ELSIF (Data_Length = 6) THEN
151           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & DATA2 & DATA3 & DATA4 & DATA5 & CRC & CRC_del & ACK_slot & ACK_del & "0" );
152         ELSIF (Data_Length = 7) THEN
153           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & DATA2 & DATA3 & DATA4 & DATA5 & DATA6 & CRC & CRC_del & ACK_slot & ACK_del
      & "0" );
154         ELSIF (Data_Length = 8) THEN
155           Frame(94 downto (95 - Frame_Length)) <= ("0" & id & RTR & "00" & control & DATA
      0 & DATA1 & DATA2 & DATA3 & DATA4 & DATA5 & DATA6 & DATA7 & CRC & CRC_del & ACK_slot &
      ACK_del & "0" );
156         END IF;
157         position := 0;
158         -- Insert the recessive and the dominant bit after every five identically
159         -- bit and transmit with bit encoding Non Return to Zero.
160         FOR i IN 0 TO 94 LOOP
161         position := position+1;      -- Count the position up
162           IF (Frame(i)='1') THEN     -- Compare the frame bit with bit value one
163             count1 := count1+1;      -- Count the value count1 up with one
164             count0 := 0;             -- Set the count0 to zero
165             IF (count1 > 5) THEN     -- If Count is 5 the dominant bit will be set
166               count1 := 0;           -- Set the count1 to zero
167               count0 := 1;           -- Set the count0 to one
168               Frame((95-position-1) downto (95-Frame_Length-position-1)) <= ((95-position
      ) downto (95 - Frame_Length-position));
169               Frame(i) <= '0';       -- Set the dominant bit
170               position := position+1; -- Add one to position
171             END IF;
172           ELSIF (Frame(i)='0') THEN -- Compare the frame bit with bit value zero
173             count0 := count0+1;      -- Count the value count0 up with one
174             count1 := 0;             -- Set the count1 to zero
175             IF (count0 > 5) THEN     -- If Count is 5 the recessive bit will be set
176               count0 := 0;           -- Set the count0 to zero
177               count1 := 1;           -- Set the count1 to one
178               Frame((95-position-1) downto (95-Frame_Length-position-1)) <= ((95-position
      ) downto (95 - Frame_Length-position));
179               Frame(i) <= '1';       -- Set the recessive bit
180               position := position+1; -- Add one to position
181             END IF;
182           END IF;
183         END LOOP;
184         ELSE Frame <= Frame(93 DOWNTO 0) & '1';
185         END IF;
186       END IF;
187     END PROCESS;
188     dout <= Frame(94); -- Send data out on the CAN TX port
189   end CAN_TX;
```

Figures 6.8 – VHDL code for transmit data via Data Link Layer.

6.4. Simulation

This simulation in figure 6.9 shows the data output from the CAN transmission VHDL code there will be send out to the CAN hardware interface from the FPGA.

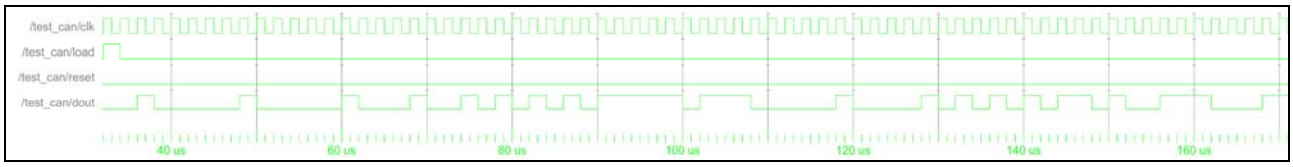


Figure 6.9 – Simulation of CAN TX in ModelSim

6.5. Test and result

The CAN transmission is tested by sending the data packet as described in chapter 6.3 the data packet is received with a CAN-USB unit from www.canusb.com. The CAN unit is connected to the computer via USB and as a node at the network. For measurement the right data packet in the development process is connected an Oscilloscope from the company Tektronix type TDS 220 there are a digital real-time stores oscilloscope.

There is measurement differential on the CAN bus signal, between the CAN_L (pin2) and the CAN_H (pin 7) at the SUB-D connector. The ground probe from the oscilloscope is connected to pin 2, the signal is inverted compared with the signal shown in figure 6.7.

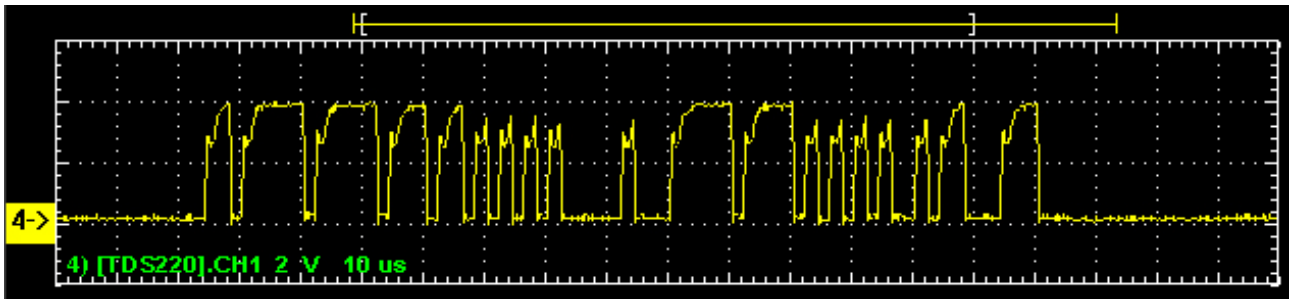


Figure 6.10 – Oscilloscope picture from transmission of CAN data packet.

Chapter 7:

Software Setup

7.1. Introduction

This chapter is a short guide to setup the software and give an overview of the step for step development of a program in VHDL and C language for a PicoBlaze microcontroller ready to run in a Spartan-3 FPGA from Xilinx.

Download the Xilinx Design tool, Project Navigator, ISE WebPACK Service Pack 6.3.03.i and the Simulation tool, ModelSim XE II 5.8C from xilinx.com. Install the software onto the computer and copy the project library named BGEPB1 from the CD-ROM which has been attached at the last page in this report to the rood of you computer or in a folder with less at eight character.

7.2. Setup of C and ASM Compiler

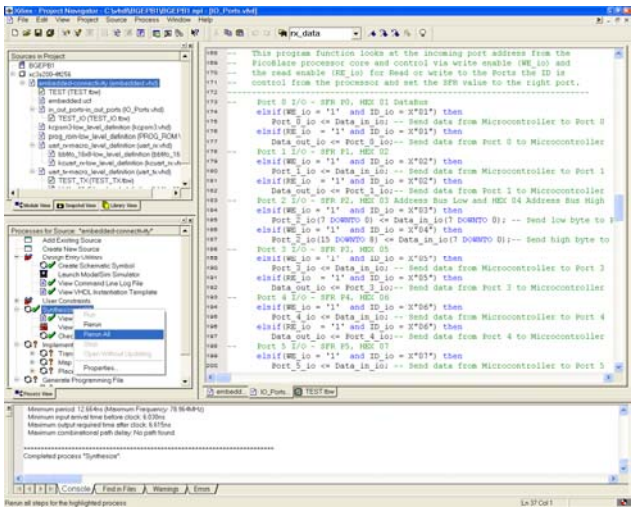


Figure 7.1 – Xilinx Project Navigator

Start the Xilinx Project Navigator and open the project from the library named BGEPB1.

Compile the project with left click on the *embedded-connectivity (embedded.vhd)* as it is marked with a blue line in the *Sources in Project window*, and after-wards right click at the *Synthesize-XST* and chose *Rerun All* in the Process for Source window.

Minimize the Project Navigator and copy the PicoBlaze C compiler named PCCOMP from the CD-ROM to the C:\ root of the computer. Open the Notepad document named prog_rom.c and edit in the document from the CD if you wish to change something otherwise just save it and minimize the document. Right click at the file named RUN.bat and create a shortcut to the desktop. Right click at the icon and rename it to *Compile C to ASM code* and afterwards chose Edit to change the location in the second line, where it is described where the compiled file shall be copied to “copy c:\pccomp\prog_rom.psm c:\vhdl\BGEPB1\prog_rom.psm” save and close the document and double-click on the icon named “Compile C to ASM code”. The C code will be compiled to ASM code with the PCCOMP compiler and copy to the BGEPB1 library. Show in figure 7.2 if there are errors

in the code, they will be listed numbered line-codes it will be shown in the DOS shell along with the syntax problem.

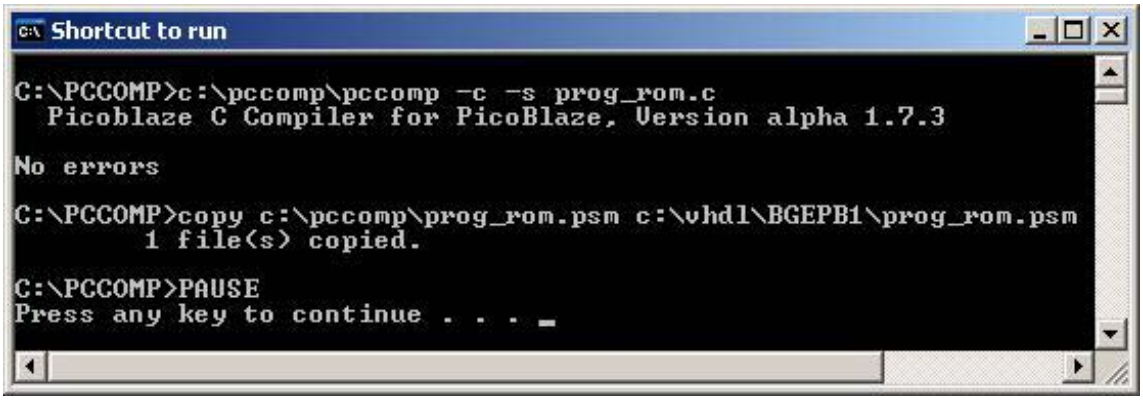


Figure 7.2 – PCCOMP C compiler run from DOS shell.

Open the library named BGEPB1 and make a shortcut to the file named RUN.bat and copy this shortcut to the desktop. Rename the ICON to "*Compile ASM to VHDL Format*" Right click and chose Edit and change the location if it is different. Double-click on the icon *Compile ASM to VHDL Format* the program compile the ASM code to machine code in VHDL format with help of the PicoBlaze compiler named KCPSM3 showed in figure 7.3. If there are any errors in the ASM code the errors be list in the DOS shell.

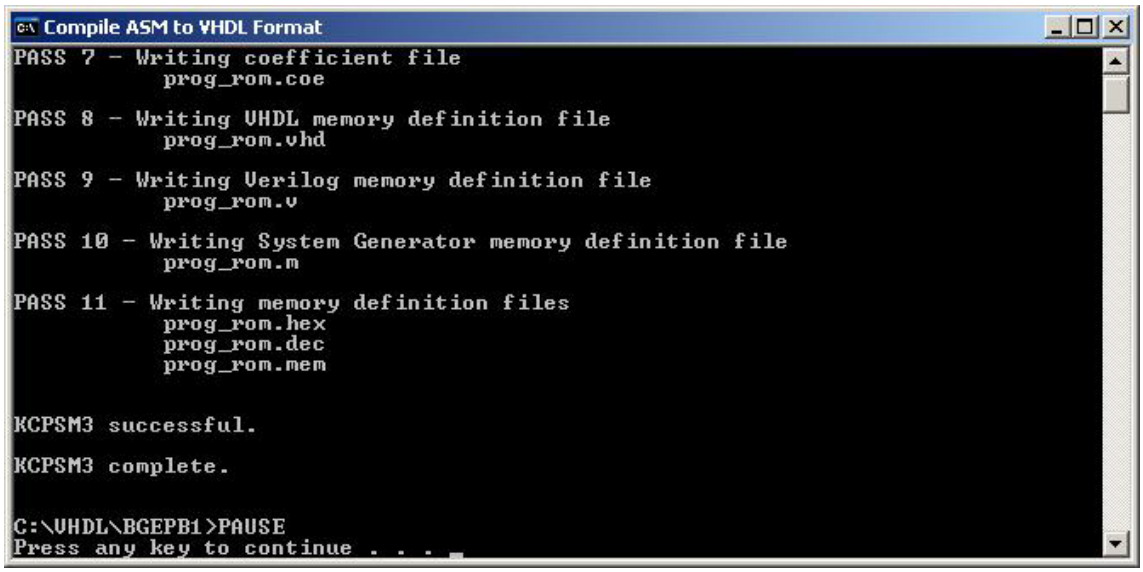


Figure 7.3 –KCPSM3 ASM compiler run from DOS shell.

Recompile the project in Xilinx project Navigator and the project is ready to be tested in ModelSim. Every time the c-code is changed it is necessary to *Compile C to ASM code* afterwards *Compile ASM to VHDL Format* and recompile the project in Xilinx Project Navigator.

7.3. Simulation in ModelSim

The easiest and most undemanding way to simulate in ModelSim is by adding a Test Bench Waveform to the project showed in this part. The description of how to setup a Test Bench mark and how to use it with ModelSim is explained in this section.

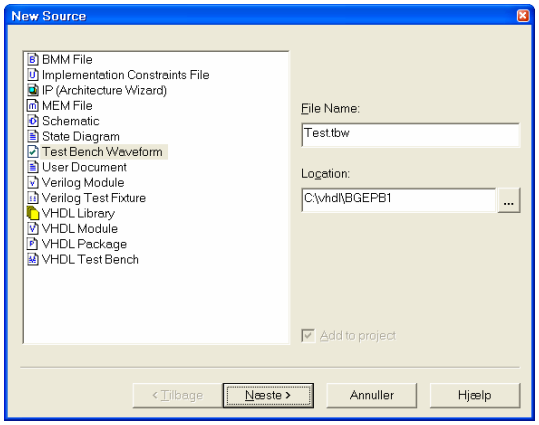


Figure 7.4 – Add New Test Bench Waveform

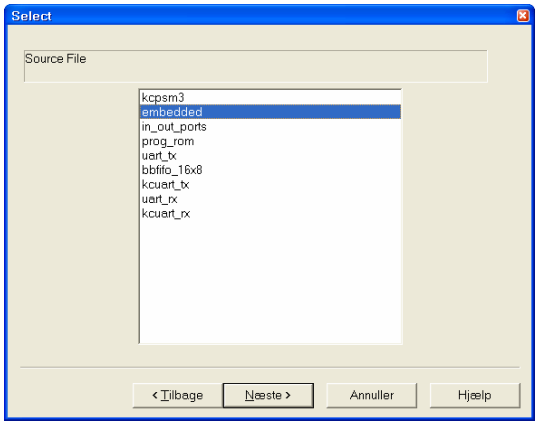


Figure 7.5 – Chose Source File for Test Bench

To add a Test Bench Waveform in Xilinx Project Navigator, right click in the *Sources in Project* window and chose *New Source*. Select *Test Bench Waveform* in the menu to the left show in figure 7.4 and enter a name for the file. Click next and chose the VHDL file you would like to test In- and Output on, in this example the embedded file show in figure 7.5 Click next and chose create.

The Project Navigator will open a window like the one which is shown in figure 7.6 in the bottom of the page. And you will be asked about clock frequency, in this case it is set to 20nS and a duty cycle at 50% because this is the speed the Spartan-3 board runs at. The blue colour shows output and yellow shows input. The reset is set high in the beginning of the simulation and afterwards it is low.

For start simulation with ModelSim double-click on the “Simulation Behavioral Model” and the program will start and run afterwards simulation for the VHDL project.

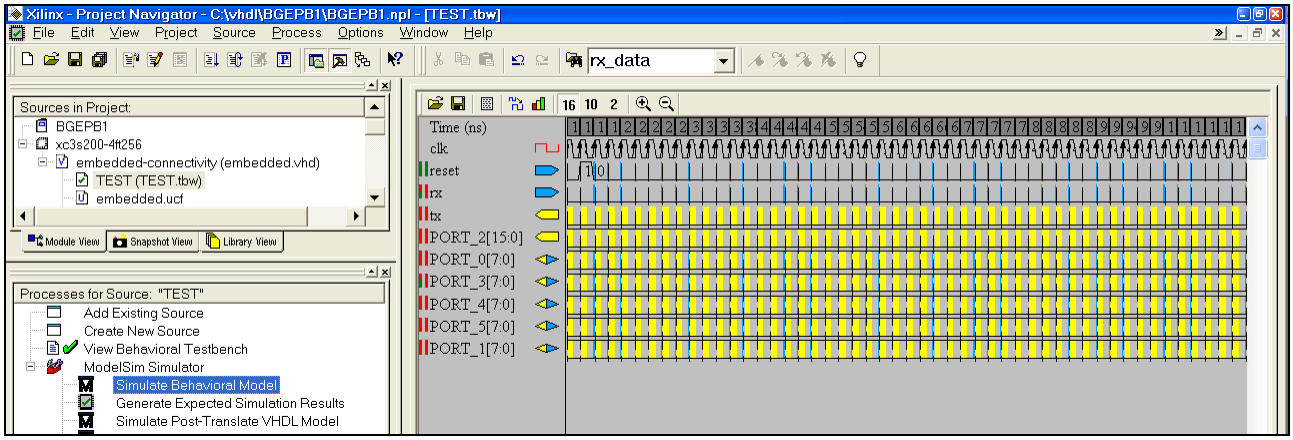


Figure 7.6 - Test Bench Waveform.

7.4. Download to FPGA via iMPACT tool

After complete compiling of the VHDL project without errors in Xilinx Project Navigator and the pin assignment is done, the project ready to be downloaded to the FPGA. But if it is the first time the project is downloaded to the FPGA there are some settings that needs to be checked, but only once.

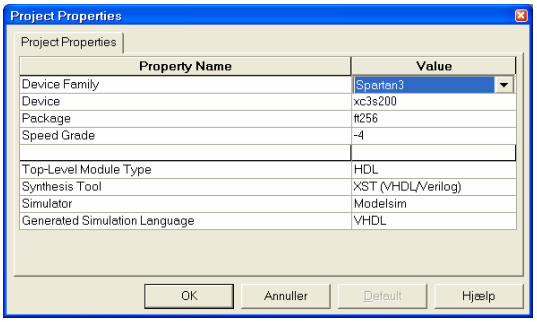


Figure 7.7 – Project Properties

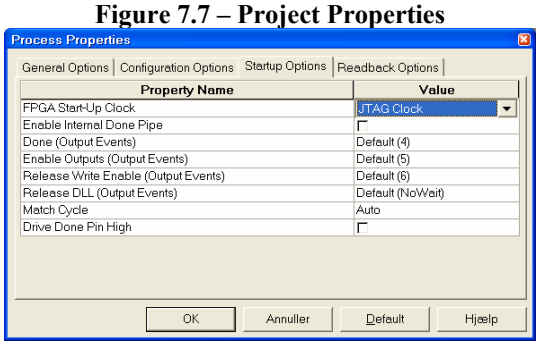


Figure 7.8 – Process Properties

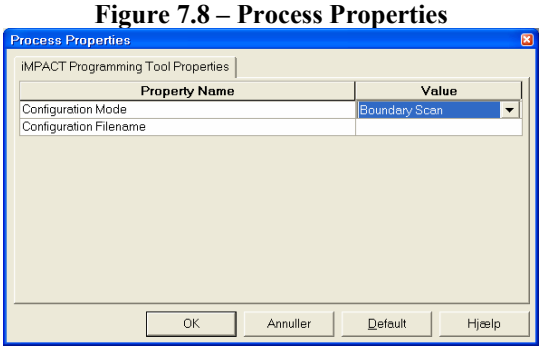


Figure 7.9 – Process Properties

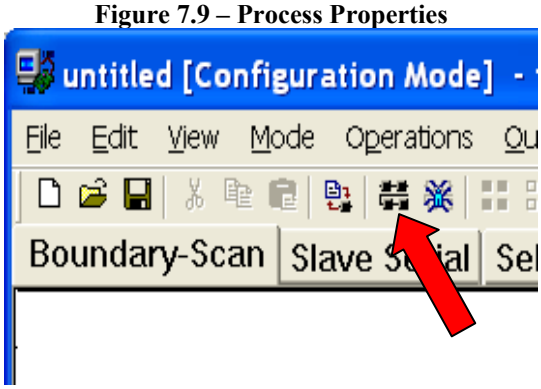


Figure 7.10 – iMPACT

Right click on the project device in the *Sources in Project* window and chose properties. In the *Project Properties* check the right Device Family, Device, Package and speed Grade is chosen as the same as the FPGA device which is used on the development board.

Left click with the mouse on the VHDL project source file in the *Sources in project* window, and chose in the *Processes for Source* window the *Generate Programming File* and right click here and chose properties in the menu. The *Process Properties* window will appear, and then chose the menu named *Startup Option* and select the function named *FPGA Start-Up Clock* to *JTAG Clock* press OK! As showed in figure 7.8.

Right click on the Configure Device (iMPACT) and chose properties in the menu. Select the Configuration Mode and chose this to Boundary Scan. Press OK! For save the change, show in figure 7.9.

Right click on the *Generate Programming File* and select *Rerun all* in the menu. Make assured that there are no warnings or errors in the compiled code. Doublet click on the *Configure Device (iMPACT).* After the new program is open *iMPACT* chose the function named Boundary-Scan in the menu-bar show in figure 7.10.

Cancel the automatic saving of files from the VHDL project, the program will automatically ask when it starts up. The Boundary-Scan has found two devices the XC3S200 FPGA and the XCF02S Flash mounted on the development Spartan-3 Starter Kit Board. If there is used another board the Boundary-Scan will via the JTAG connection find these devices there are mount on this development board.

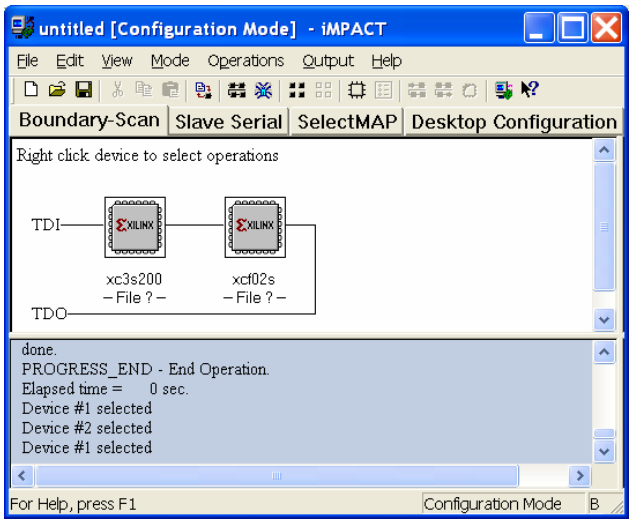


Figure 7.11 – iMPACT Boundary-Scan

The search result for the Spartan-3 Starter Kit Board is show in figure 7.11.

Right click on the FPGA XC3S200 device in the program and chose *Assign New Confirmation File*. Select the *embedded.bit* file in the library named BGEPB1 and chose open.

Right click on the FPGA device XC3S200 and chose *Program* and press *OK* for accepts programming of the device.

Note. Make sure the Jumper JP1 is removed on the development Spartan-3 Starter Kit Board for program the FPGA.

Chapter 8

Conclusion

The project has more or less been successful in reaching the aim of this project. The development of the PicoBlaze microprocessor core running in a new version of microcontroller, named BGEPB1 created with a simplified Special Function Register whit controlled parallels I/O ports, serial UART, timer and interrupts, is complete as seen in the test results.

The project period compared with the time plan has not really been fulfilling. After project week 16 where the development of the CAN bus started and the problem with programming in VHDL started for real, a lot of data converting and manipulating of data vector this have given a lot of synthesize problems In the Xilinx Project Navigator. This due to the project not just having an Implementation of a microprocessor in a FPGA but also there has been a new VHDL language to learn to be able to make the project.

Implementation of PicoBlaze Core:

The status for the Implementation of the PicoBlaze microprocessor in the microcontroller, the BGEPB1, is complete in regards to the requirement given. The microcontroller is ready to be used and it is easy to implement new function in the VHDL code e.g. more timers and extra interrupts. The only thing which has not been tested and made is a C- or ASM code example for the Interrupt control which reads the Interrupt and automatically sends an Interrupt acknowledge after end reading.

Design of CAN bus:

The CAN bus interface is made in hardware and tested in transmit- and receive-mode and tested functional. The lower Data Link Layer is made In VHDL controlling the 8-bit data packets which is send serial out with the encoding standard, known as Non Return to Zero, and inset the recessive and the dominant bit after every five identically bit. The CAN bus is not finished. The development is still missing functions as Cyclic Redundancy Check calculation and Error bit control.

The future development at the project:

The future plans for the project is to continue the development of the CAN bus interface and implementation this in the BGEPB1 microcontroller which will be available at the homepage www.bg-elektronik/fpga

_____

Benjamin Grydehoej

Related Materials and References

References:

1. **Roman – Jones, Inc. – Emulate 8051 Microprocessor in PicoBlaze IP Core**
   - http://www.roman-jones.com/PB8051Microcontroller.htm
2. **Xilinx.com - PicroBlaze 8-bit Embedded Microcontroller User Guide, Page 13-14**
   - http://www.xilinx.com/bvdocs/userguides/ug129.pdf
3. **Xilinx.com – UART Transmitter and Receiver Macros, Page 3**
   - http://www.xilinx.com/bvdocs/appnotes/xapp223.pdf
4. **Maxim-ic.com – Data sheet, Low Supply Current CAN Transceiver, page 1 & 4**
   - http://pdfserv.maxim-ic.com/en/ds/MAX3053.pdf

Bibliography:

Circuit Design with VHDL – Volnei A. Pedroni – ISBN 0-262-16224-5

Microcomputer Components - 8-Bit single-Chip Family – Siemens – User's Manual 8/95

PicoBlaze 8-bit Embedded Microcontroller User Guide – UG129 (v1.1) June 10, 2004

Xilinx's homepage - http://www.xilinx.com/bvdocs/userguides/ug129.pdf

PicoBlaze C compiler User's Manual 1.1 July 2005 – Francesco Poderico

Francesco Poderico's homepage - http://www.poderico.co.uk

Spartan-3 Starter Kit Board User Guide – UG130 (v1.1) May 13, 2005

Xilinx's homepage - http://www.xilinx.com/bvdocs/userguides/ug130.pdf

Software:

Xilinx Design tool - Project Navigator - ISE Service Pack 6.3.03i (Windows)

Xilinx's homepage - http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp

Simulation program - ModelSim XE II/Starter 5.8C (Windows)

Xilinx's homepage - http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp

PicoBlaze C compiler – PCCOMP (DOS)

Francesco Poderico's homepage - http://www.poderico.co.uk/down.html

PicoBlaze Assembler compiler – KCPSM3 (DOS)

Xilinx's homepage -
http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sGlobalNavPick=PRODUCTS&sSecondaryNavPick=Design+Tools&key=picoblaze-S3-V2-Pro

PicoBlaze Debugger – pBlazIDE (Windows)

Xilinx's homepage - Mediatronix's homepage - http://www.mediatronix.com/pBlazeIDE.htm

The VHDL code for I/O Interface

C:\vhdl\BGEPB1\IO_Ports.vhd

```
1   --*************************************************************************
2   --*COPYRIGHT: BENJAMIN GRYDEHOEJ - WWW.BG-ELEKTRONIK.DK - 2006 - FPGA SPARTAN-3*
3   --*************************************************************************
4   --  Author:          Benjamin Grydehoej
5   --  Create the:      11th November, 2005
6   --  Last update the: 11th April, 2006
7   --  File:            IO_PORTS.VHD
8   --  Target Hardware: Xilinx Spartan3 - XC3S200
9   --  Tool chain:      Xilinx - Project Navicator 6.3.03i
10  --  Version:         1.0.C
11  --
12  --  DESCRIPTION:
13  --  ============
14  --  This VHDL code controls the Input/Output Ports, Serial UART and the SFR
15  --  register with Interrupt controls and Timer function. The Sub-functions
16  --  is described in the code before the program. All commandos for
17  --  communication in this program are control by the Special Function Register
18  --  there is listed under this text.
19  --  ====================================================
20  --              SPECIAL FUNCTION REGISTER:
21  --  ====================================================
22  --   Symbol:   Name:                            Address:
23  --   P0        Port 0                           HEX 01
24  --   P1        Port 1                           HEX 02
25  --   P2L       Port 2 Address Bus low byte      HEX 03
26  --   P2H       Port 2 Address Bus High byte     HEX 04
27  --   P3        Port 3                           HEX 05
28  --   P4        Port 4                           HEX 06
29  --   P5        Port 5                           HEX 07
30  --   SBUF      Serial channel buffer register   HEX 08
31  --   TLBS      Timer Low BAUE Rate Serial       HEX 09
32  --   THBS      Timer HIGH BAUE Rate Serial      HEX 0A
33  --   SCON      Serial channel control register  HEX 0B
34  --   IEN0      Interrupt enable register 0      HEX 0C
35  --   IEN1      Interrupt enable register 1      HEX 0D
36  --   ISC0      Interrupt service control register HEX 0E
37  --   TCON      Timer service control register   HEX 0F
38  --   TC0       Timer Count 0                    HEX 10
39  --   TCL1      Timer Count Low 1                HEX 11
40  --   TCH1      Timer Count High 1               HEX 12
41  ----------------------------------------------------------------------------
42  -- Standard IEEE libraries
43  ----------------------------------------------------------------------------
44  library IEEE;
45  use IEEE.STD_LOGIC_1164.ALL;
46  use IEEE.STD_LOGIC_ARITH.ALL;
47  use IEEE.STD_LOGIC_UNSIGNED.ALL;
48  ----------------------------------------------------------------------------
49  entity in_out_ports is
50    Port (            CLK_io : in    std_logic;
51                    Reset_io : in    std_logic;
52                       WE_io : in    std_logic;
53                       RE_io : in    std_logic;
54                       ID_io : in    std_logic_vector(7 downto 0);
55                  Data_in_io : in    std_logic_vector(7 downto 0);
56                 Data_out_io : out   std_logic_vector(7 downto 0);
57                Interrupt_io : out   std_logic;
58            Interrupt_ack_io : in    std_logic;
59             rx_data_present : in    std_logic;
60                   Port_0_io : inout std_logic_vector(7 downto 0);
61                   Port_1_io : inout std_logic_vector(7 downto 0);
62                   Port_2_io : out   std_logic_vector(15 downto 0);
63                   Port_3_io : inout std_logic_vector(7 downto 0);
64                   Port_4_io : inout std_logic_vector(7 downto 0);
65                   Port_5_io : inout std_logic_vector(7 downto 0);
66                    TBS_uart : out   std_logic_vector(15 downto 0);
67                    BDP_uart : in    std_logic;
68                    RBH_uart : in    std_logic;
```

Page: 1

C:\vhdl\BGEPB1\IO_Ports.vhd

```
69                    RBF_uart : in    std_logic;
70                    TBH_uart : in    std_logic;
71                    TBF_uart : in    std_logic;
72                write_to_uart : out   std_logic;
73                 data_to_uart : out   std_logic_vector(7 downto 0);
74               read_from_uart : out   std_logic;
75               data_from_uart : in    std_logic_vector(7 downto 0);
76                    TR0_timer : inout std_logic;
77                    TF0_timer : in    std_logic;
78                    TC0_timer : out   std_logic_vector(7 downto 0);
79                    TR1_timer : inout std_logic;
80                    TF1_timer : in    std_logic;
81                    TC1_timer : out   std_logic_vector(15 downto 0));
82
83    end in_out_ports;
84
85    ---------------------------------------------------------------------------
86    --  This program function is the handling of Input and Output control
87    --  after the Special Function Register (SFR) for read and write to
88    --  ports and option of Interrupt Service Routines and Timer control etc.
89    ---------------------------------------------------------------------------
90    architecture in_out_ports of in_out_ports is
91
92    ---------------------------------------------------------------------------
93    -- Signals for Interrupt connections
94    ---------------------------------------------------------------------------
95    -- IEN0 - Interrupt Enables 0 Variables
96    signal     EA_int : std_logic;
97    signal    ET1_int : std_logic;
98    signal    ET0_int : std_logic;
99    signal    EX2_int : std_logic;
100   signal    EX1_int : std_logic;
101   signal    EX0_int : std_logic;
102   Signal    ES0_int : std_logic;
103   signal    IC0_int : std_logic;
104   signal    IS0_int : std_logic;
105   signal    IT2_int : std_logic;
106   signal    IT1_int : std_logic;
107   signal    IT0_int : std_logic;
108   signal    IX2_int : std_logic;
109   signal    IX1_int : std_logic;
110   signal    IX0_int : std_logic;
111   -- Interrupt Variables
112   signal     X0_int : std_logic;
113   signal     X1_int : std_logic;
114   signal     X2_int : std_logic;
115
116   begin
117      process
118      begin
119
120         wait until (CLK_io'event and CLK_io='1');
121         X0_int <= Port_3_io(0); -- Read the Port 1 bit 0 value and save
122                                 -- it in the Internal signal.
123         X1_int <= Port_3_io(1); -- Read the Port 1 bit 1 value and save
124                                 -- it in the Internal signal.
125         X2_int <= Port_3_io(2); -- Read the Port 1 bit 2 value and save
126                                 -- it in the Internal signal.
127   ---------------------------------------------------------------------------
128   --    RESET:
129   --    This function set ports level after reset and
130   --    define the value for variables
131   ---------------------------------------------------------------------------
132         if(Reset_io='1') then
133            -- PORTS  level after reset
134            Port_0_io <= "ZZZZZZZZ";  -- Set Port 0 to high impedance level
135            Port_1_io <= "ZZZZZZZZ";  -- Set Port 1 to high impedance level
136            Port_2_io <= X"0000";     -- Set Port 2 to Address 0 (Hexadecimal)
```

Page: 2

C:\vhdl\BGEPB1\IO_Ports.vhd

```
137           Port_3_io <= "ZZZZZZZZ";  -- Set Port 3 to high impedance level
138           Port_4_io <= "ZZZZZZZZ";  -- Set Port 4 to high impedance level
139           Port_5_io <= "ZZZZZZZZ";  -- Set Port 5 to high impedance level
140           -- TIMER
141           TBS_uart  <= X"0146";     -- Set Baud Rate at 9600 (dec 326) HEX 0146
142           TC0_timer <= X"FF";       -- Set Timer Value to 255
143           TC1_timer <= X"FFFF";     -- Set Timer Value to 65535
144           TR0_timer <= '0';         -- Clear Timer 0 Run (Timer stop)
145           TR1_timer <= '0';         -- Clear Timer 1 Run (Timer stop)
146           -- INTERRUPT SYSTEM
147           Interrupt_io <= '0';      -- Set global Interrupt to zero
148           -- IEN0 - Interrupt Enables 0
149           EA_int     <= '0';        -- Clear Enable All Interrupts
150   --      WDT_int    <= '0';        -- Clear WDT Interrupt
151   --      ET2_int    <= '0';        -- Clear Enable Timer 2 Interrupt
152           ET1_int    <= '0';        -- Clear Enable Timer 1 Interrupt
153           ET0_int    <= '0';        -- Clear Enable Timer 0 Interrupt
154           EX2_int    <= '0';        -- Clear Enable Extern 2 Interrupt
155           EX1_int    <= '0';        -- Clear Enable Extern 1 Interrupt
156           EX0_int    <= '0';        -- Clear Enable Extern 0 Interrupt
157           -- IEN1 - Interrupt Enables 1
158   --      EC0_int    <= '0';        -- Clear Enable CAN Interrupt
159           ES0_int    <= '0';        -- Clear Enable Serial Interrupt
160           -- ISC0 - Interrupt Service Control
161           IC0_int    <= '0';        -- Clear FLAG CAN-BUS Interrupt
162           IS0_int    <= '0';        -- Clear FLAG Serial Interrupt
163           IT2_int    <= '0';        -- Clear FLAG Timer Overflow 2 Interrupt
164           IT1_int    <= '0';        -- Clear FLAG Timer Overflow 1 Interrupt
165           IT0_int    <= '0';        -- Clear FLAG Timer Overflow 0 Interrupt
166           IX2_int    <= '0';        -- Clear FLAG Extern 2 Interrupt
167           IX1_int    <= '0';        -- Clear FLAG Extern 1 Interrupt
168           IX0_int    <= '0';        -- Clear FLAG Extern 0 Interrupt
169
170   ------------------------------------------------------------------------
171   --   Input & Output Interface:
172   --   This program function looks at the incoming port address from the
173   --   PicoBlaze processor core and control via write enable (WE_io) and
174   --   the read enable (RE_io) for Read or write to the Ports the ID is
175   --   control from the processor and set the SFR value to the right port.
176   ------------------------------------------------------------------------
177   --    Port 0 I/O - SFR P0, HEX 01 DataBus
178         elsif(WE_io = '1'  and ID_io = X"01") then
179            Port_0_io <= Data_in_io; -- Send data from Microcontroller to Port 0
180         elsif(RE_io  = '1' and ID_io = X"01") then
181            Data_out_io <= Port_0_io;-- Send data from Port 0 to Microcontroller
182   --    Port 1 I/O - SFR P1, HEX 02
183         elsif(WE_io = '1'  and ID_io = X"02") then
184            Port_1_io <= Data_in_io; -- Send data from Microcontroller to Port 1
185         elsif(RE_io  = '1' and ID_io = X"02") then
186            Data_out_io <= Port_1_io;-- Send data from Port 1 to Microcontroller
187   --    Port 2 I/O - SFR P2, HEX 03 Address Bus Low and HEX 04 Address Bus High
188         elsif(WE_io = '1'  and ID_io = X"03") then
189            Port_2_io(7 DOWNTO 0) <= Data_in_io(7 DOWNTO 0); -- Send low byte to Port 2
190         elsif(WE_io = '1'  and ID_io = X"04") then
191            Port_2_io(15 DOWNTO 8) <= Data_in_io(7 DOWNTO 0);-- Send high byte to Port 2
192   --    Port 3 I/O - SFR P3, HEX 05
193         elsif(WE_io = '1'  and ID_io = X"05") then
194            Port_3_io <= Data_in_io; -- Send data from Microcontroller to Port 3
195         elsif(RE_io  = '1' and ID_io = X"05") then
196            Data_out_io <= Port_3_io;-- Send data from Port 3 to Microcontroller
197   --    Port 4 I/O - SFR P4, HEX 06
198         elsif(WE_io = '1'  and ID_io = X"06") then
199            Port_4_io <= Data_in_io; -- Send data from Microcontroller to Port 4
200         elsif(RE_io  = '1' and ID_io = X"06") then
201            Data_out_io <= Port_4_io;-- Send data from Port 4 to Microcontroller
202   --    Port 5 I/O - SFR P5, HEX 07
203         elsif(WE_io = '1'  and ID_io = X"07") then
204            Port_5_io <= Data_in_io; -- Send data from Microcontroller to Port 5
```

Page: 3

C:\vhdl\BGEPB1\IO_Ports.vhd

```
205         elsif(RE_io  = '1' and ID_io = X"07") then
206            Data_out_io <= Port_5_io;-- Send data from Port 5 to Microcontroller
207   ------------------------------------------------------------------------
208   --  SERIAL DATA CONTROL:
209   --  This function controls the Serial UART with write and read via SBUF and
210   --  the option for the BAUD rate timing there control the speed for the serial
211   --  communication. The Serial Channel Control register SCON do it possible to
212   --  read the status flag for the received and transmit buffer and look after
213   --  the status flag BDP for receive data.
214   ------------------------------------------------------------------------
215   --    SBUF Read and write to Comport - SFR SBUF, HEX 08
216         elsif(WE_io = '1'  and ID_io = X"08") then -- Write to Serial Buffer
217            write_to_uart <= WE_io;         -- Enable write to UART
218            data_to_uart  <= Data_in_io;    -- Send Data to the UART buffer from
219                                            -- the SFR register named SBUF
220         elsif(RE_io  = '1' and ID_io = X"08") then -- Read to Serial Buffer
221            read_from_uart <= RE_io;        -- Enable read to UART
222            Data_out_io <= data_from_uart; -- Read Data from the UART buffer to
223                                            -- the SFR register named SBUF
224   --    Timer Baud rate serial, low byte - SFR TLBS, HEX 09
225         elsif(WE_io = '1'  and ID_io = X"09") then -- Write the low byte
226            TBS_uart(7 DOWNTO 0) <= Data_in_io; -- Send the low data byte to the Timer
227                                               -- Baud rate Serial for BAUD rate timing
228   --    Timer Baud rate serial, high byte - SFR THBS, HEX 0A
229         elsif(WE_io  = '1' and ID_io = X"0A") then -- Write the high byte
230            TBS_uart(15 DOWNTO 8) <= Data_in_io;-- Send the high data byte to the Timer
231                                               -- Baud rate Serial for BAUD rate timing
232
233   --    Serial Channel Control Register - SFR SCON, HEX 0B
234   --     -------------------------------------------
235   --    |  X  |  X  |  X  | BDP | RBH | RBF | TBH | TBF |
236   --     -------------------------------------------
237         elsif(RE_io  = '1' and ID_io = X"0B") then  -- Read the status flag from Serial
238                                                    -- Channel Control Register SCON
239            Data_out_io <= ("000" & BDP_uart & RBH_uart & RBF_uart & TBH_uart & TBF_uart);
240
241   -- Serial Interrupt handling
242         elsif (ES0_int='1' and IS0_int='0' and rx_data_present='1' and EA_int='1') then
243           IS0_int <= '1'; -- Set Serial Interrupt FLAG
244
245   ------------------------------------------------------------------------
246   --  INTERRUPT SYSTEM:
247   --  This program handle the Interrupt System there use the three register
248   --  named IEN0 - Interrupt Enable 0, IEN1 - Interrupt Enable 2 and ISC0 -
249   --  Interrupt Service Control. Activate with help of the SFR, the IEN0
250   --  and IEN1 enables the interrupt and the ISC0 show the status for the
251   --  interrupts.
252   ------------------------------------------------------------------------
253   --    INTERRUPT ENABLES - SFR IEN0, HEX 0C
254   --     -----------------------------------------------
255   --    | EA  | WDT | ET2 | ET1 | ET0 | EX2 | EX1 | EX0 |
256   --     -----------------------------------------------
257         elsif(WE_io  = '1' and ID_io = X"0C") then
258            EA_int <= Data_in_io(7);  -- Activate or deactivate all Interrupts EA
259   --       WDT_int <= Data_in_io(6);  -- Activate or deactivate WDT
260   --       ET2_int <= Data_in_io(5);  -- Activate or deactivate Interrupts Timer 2
261            ET1_int <= Data_in_io(4);  -- Activate or deactivate Interrupts Timer 1
262            ET0_int <= Data_in_io(3);  -- Activate or deactivate Interrupts Timer 0
263            EX2_int <= Data_in_io(2);  -- Activate or deactivate External Interrupt 2
264            EX1_int <= Data_in_io(1);  -- Activate or deactivate External Interrupt 1
265            EX0_int <= Data_in_io(0);  -- Activate or deactivate External Interrupt 0
266   --    INTERRUPT ENABLES - SFR IEN1, HEX 0D
267   --     -----------------------------------------------
268   --    |  X  |  X  |  X  |  X  |  X  |  X  | EC0 | ES0 |
269   --     -----------------------------------------------
270         elsif(WE_io  = '1' and ID_io = X"0D") then
271   --       EC0_int <= Data_in_io(1);  -- Activate or deactivate CAN-BUS Interrupt
272            ES0_int <= Data_in_io(0);  -- Activate or deactivate Serial Interrupt
```

Page: 4

C:\vhdl\BGEPB1\IO_Ports.vhd

```
273
274   --    INTERRUPT SERVICE CONTROL - SFR ISC0, HEX 0E
275   --     ------------------------------------------------
276   --    | IC0 | IS0 | IT2 | IT1 | IT0 | IX2 | IX1 | IX0 |
277   --     ------------------------------------------------
278         elsif(WE_io  = '1' and ID_io = X"0E") then
279            IC0_int <= Data_in_io(7);  -- Clear Interrupt FLAG for CAN-BUS
280            IS0_int <= Data_in_io(6);  -- Clear Interrupt FLAG for Serial
281            IT2_int <= Data_in_io(5);  -- Clear Interrupt FLAG for Timer 2
282            IT1_int <= Data_in_io(4);  -- Clear Interrupt FLAG for Timer 1
283            IT0_int <= Data_in_io(3);  -- Clear Interrupt FLAG for Timer 0
284            IX2_int <= Data_in_io(2);  -- Clear Interrupt FLAG for External 2
285            IX1_int <= Data_in_io(1);  -- Clear Interrupt FLAG for External 1
286            IX0_int <= Data_in_io(0);  -- Clear Interrupt FLAG for External 0
287         elsif(RE_io  = '1' and ID_io = X"0E") then
288            Data_out_io <= (IC0_int & IS0_int & IT2_int & IT1_int & IT0_int & IX2_int & IX
    1_int & IX0_int);
289   --    External Interrupt Service Routine
290         elsif (EX0_int='1' and IX0_int='0' and X0_int='1' and EA_int='1') then
291           IX0_int <= '1';        -- Set Interrupt FLAG
292           Interrupt_io <= '1';  -- Send Interrupt to PicoBlaze
293         elsif (EX1_int='1' and IX1_int='0' and X1_int='1' and EA_int='1') then
294           IX1_int <= '1';        -- Set Interrupt FLAG
295           Interrupt_io <= '1';  -- Send Interrupt to PicoBlaze
296         elsif (EX2_int='1' and IX2_int='0' and X2_int='1' and EA_int='1') then
297           IX2_int <= '1';        -- Set Interrupt FLAG
298           Interrupt_io <= '1';  -- Send Interrupt to PicoBlaze
299   ------------------------------------------------------------------------------
300   --  This program function support the Timer option via the SFR TCON there
301   --  start/stop timer to run and the reading flag function for timer interrupt.
302   --  The supports also the reload value for Timer 0 and Timer 2.
303   ------------------------------------------------------------------------------
304   --    TIMER SERVICE CONTROL - SFR TCON, HEX 0F
305   --     ------------------------------------------------
306   --    |  X  |  X  | TF2 | TR2 | TF1 | TR1 | TF0 | TR0 |
307   --     ------------------------------------------------
308         elsif(WE_io  = '1' and ID_io = X"0F") then  -- Write to TCON register
309                                                    -- for Start/Stop
310            TR0_timer <= Data_in_io(0);             -- Start timer 0 with set a '1'
311            TR1_timer <= Data_in_io(2);             -- Start timer 1 with set a '1'
312         elsif(RE_io  = '1' and ID_io = X"0F") then  -- Read status flag and which timer
313                                                    -- there are on!
314            Data_out_io <= ("0000" &  TF1_timer & TR1_timer & TF0_timer & TR0_timer);
315
316   --    Timer Count 0 - TC0, HEX 10
317         elsif(WE_io  = '1' and ID_io = X"10") then  -- Reload new value to timer 0
318            TC0_timer <= Data_in_io;
319   --    Timer Count 1 - TCL1, HEX 11 and TCH1 HEX 12
320         elsif(WE_io  = '1' and ID_io = X"11") then  -- Reload new value to Timer 2
321            TC1_timer(7 DOWNTO 0) <= Data_in_io;     -- Set the low byte
322         elsif(WE_io  = '1' and ID_io = X"12") then
323            TC1_timer(15 DOWNTO 8) <= Data_in_io;    -- Set the high byte
324
325   --    Timer Interrupt Service Routine
326         elsif (ET0_int='1' and TF0_timer='1' and IT0_int='0' and EA_int='1') then
327           IT0_int <= '1';       -- Set Timer 0 Interrupt FLAG
328           Interrupt_io <= '1'; -- Send Interrupt to PicoBlaze
329         elsif (ET1_int='1' and TF1_timer='1' and IT1_int='0' and EA_int='1') then
330           IT1_int <= '1';       -- Set Timer 1 Interrupt FLAG
331           Interrupt_io <= '1'; -- Send Interrupt to PicoBlaze
332
333   --    Relase Interrupt
334         elsif (Interrupt_ack_io = '1') then -- When Interrupt ACK is active
335           Interrupt_io <= '0'; -- Set global Interrupt to zero
336
337         end if;
338      end process;
339   end in_out_ports;
```

Page: 5

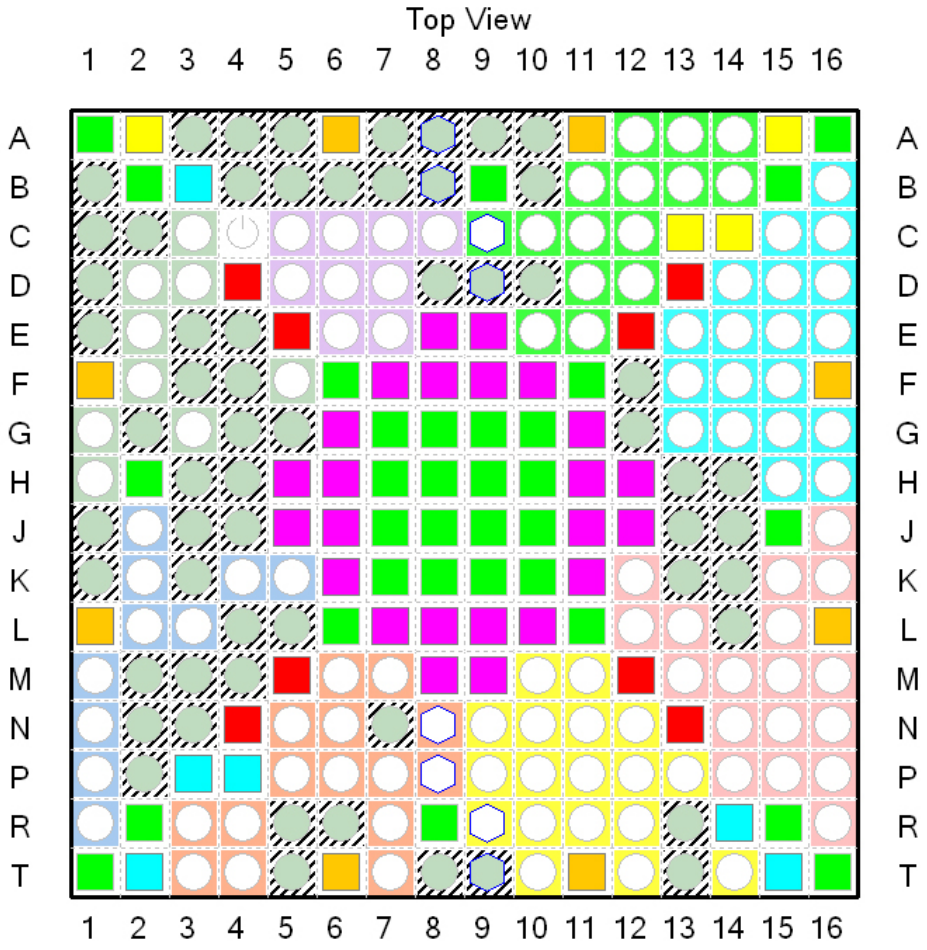
Special Function Register (BGEPB1.h)

```
//*************************************************************************************************
//*COPYRIGHT: BENJAMIN GRYDEHOEJ - WWW.BG-ELEKTRONIK.DK - 2006 - SFR for BGEPB1 *
//*************************************************************************************************
// Author:                  Benjamin Grydehoej
// Create the:              4th February, 2006
// Last update the:         14th April, 2006
// File:                    BGEPB1.h
// Target Hardware:         Xilinx Spartan3 - XC3S200
// Tool chain:              Notepad - Microsoft Version 5.1
// Compiler:                PCCOMP alpha 1.7.3 by Francesco Poderico
// Version:                 1.0.A
//
// Special Function Register for BGEPB1
//*************************************************************************************************

// Parallel port ID:
#define P0                  0x01 // Port 0 8-bit  I/O - SFR P0, HEX 01 DataBus
#define P1                  0x02 // Port 1 8-bit  I/O - SFR P1, HEX 02 Data I/O
#define P2L                 0x03 // Port 2 8-bit    O - SFR P2L, Low  byte HEX 03 AddressBus
#define P2H                 0x04 // Port 2 8-bit    O - SFR P2H, high byte HEX 04 AddressBus
#define P3                  0x05 // Port 3 8-bit  I/O - SFR P3, HEX 04 Data I/O
#define P4                  0x06 // Port 4 8-bit  I/O - SFR P4, HEX 05 Data I/O
#define P5                  0x07 // Port 5 8-bit  I/O - SFR P5, HEX 06 Data I/O

// Serial Data:
#define SBUF                0x08 // Serial Buffer
#define TLBS                0x09 // Timer Baud rate serial, low byte  - SFR TLBS, HEX 09
#define THBS                0x0A // Timer Baud rate serial, high byte - SFR TLBS, HEX 10
#define SCON                0x0B // Serial Channel Control Register   - SFR SCON, HEX 0B


// Interrupt Service Rutine:
#define IEN0                0x0C // INTERRUPT ENABLES - SFR IEN0, HEX 0C
#define IEN1                0x0D // INTERRUPT ENABLES - SFR IEN1, HEX 0C
#define ISC0                0x0E // INTERRUPT SERVICE CONTROL - SFR ISC0, HEX 0E

// Timer Service Rutine:
#define TCON                0x0F // TIMER SERVICE CONTROL - SFR TCON, HEX 0F
#define TC0                 0x10 // Timer Count 0 - TC0, HEX 10
#define TCL1                0x11 // Timer Count 1 - Low byte TCL1 HEX 11
#define TCH1                0x12 // Timer Count 1 - High byte TCH1 HEX 12
```

Pin Option for FPGA and Development board

I/O Name:	I/O Direction	PIN:	Bank:	Connector:	SRAM:
tx	Output	R13	BANK4	TXD	
rx	Input	T13	BANK4	RXD	
reset	Input	L14	BANK3	BTN3 (User Reset)	
PORT_5<7>	InOut	A10	BANK1	A2 Expansion Connector - 28	
PORT_5<6>	InOut	B10	BANK1	A2 Expansion Connector - 27	
PORT_5<5>	InOut	A9	BANK1	A2 Expansion Connector - 26	
PORT_5<4>	InOut	A8	BANK0	A2 Expansion Connector - 25	
PORT_5<3>	InOut	B8	BANK0	A2 Expansion Connector - 24	
PORT_5<2>	InOut	A7	BANK0	A2 Expansion Connector - 23	
PORT_5<1>	InOut	B7	BANK0	A2 Expansion Connector - 22	
PORT_5<0>	InOut	B6	BANK0	A2 Expansion Connector - 21	
PORT_4<7>	InOut	A5	BANK0	A2 Expansion Connector - 20	
PORT_4<6>	InOut	B5	BANK0	A2 Expansion Connector - 19	
PORT_4<5>	InOut	A4	BANK0	A2 Expansion Connector - 18	
PORT_4<4>	InOut	B4	BANK0	A2 Expansion Connector - 17	
PORT_4<3>	InOut	A3	BANK0	A2 Expansion Connector - 16	
PORT_4<2>	InOut	D10	BANK1	A2 Expansion Connector - 15	
PORT_4<1>	InOut	D9	BANK1	A2 Expansion Connector - 14	
PORT_4<0>	InOut	D8	BANK0	A2 Expansion Connector - 13	
PORT_3<7>	InOut	K13	BANK3	Slider Switch (SW7)	
PORT_3<6>	InOut	K14	BANK3	Slider Switch (SW6)	
PORT_3<5>	InOut	J13	BANK3	Slider Switch (SW5)	
PORT_3<4>	InOut	J14	BANK3	Slider Switch (SW4)	
PORT_3<3>	InOut	H13	BANK2	Slider Switch (SW3)	
PORT_3<2>	InOut	H14	BANK2	Slider Switch (SW2)	
PORT_3<1>	InOut	G12	BANK2	Slider Switch (SW1)	
PORT_3<0>	InOut	F12	BANK2	Slider Switch (SW0)	
PORT_2<15>	Output	K3	BANK6	A1 Expansion Connector - 34	A15
PORT_2<14>	Output	J3	BANK6	A1 Expansion Connector - 31	A14
PORT_2<13>	Output	J4	BANK6	A1 Expansion Connector - 32	A13
PORT_2<12>	Output	H4	BANK7	A1 Expansion Connector - 29	A12
PORT_2<11>	Output	H3	BANK7	A1 Expansion Connector - 30	A11
PORT_2<10>	Output	G5	BANK7	A1 Expansion Connector - 27	A10
PORT_2<9>	Output	E4	BANK7	A1 Expansion Connector - 28	A9
PORT_2<8>	Output	E3	BANK7	A1 Expansion Connector - 25	A8
PORT_2<7>	Output	F4	BANK7	A1 Expansion Connector - 26	A7
PORT_2<6>	Output	F3	BANK7	A1 Expansion Connector - 23	A6
PORT_2<5>	Output	G4	BANK7	A1 Expansion Connector - 24	A5
PORT_2<4>	Output	L4	BANK6	A1 Expansion Connector - 14	A4
PORT_2<3>	Output	M3	BANK6	A1 Expansion Connector - 12	A3
PORT_2<2>	Output	M4	BANK6	A1 Expansion Connector - 10	A2
PORT_2<1>	Output	N3	BANK6	A1 Expansion Connector - 8	A1
PORT_2<0>	Output	L5	BANK6	A1 Expansion Connector - 6	A0

I/O Name:	I/O Direction	PIN:	Bank:	Connector:	SRAM:
PORT_1<7>	InOut	B1	BANK7	A1 Expansion Connector - 19	
PORT_1<6>	InOut	C1	BANK7	A1 Expansion Connector - 17	
PORT_1<5>	InOut	C2	BANK7	A1 Expansion Connector - 15	
PORT_1<4>	InOut	R5	BANK5	A1 Expansion Connector - 13	
PORT_1<3>	InOut	T5	BANK5	A1 Expansion Connector - 11	
PORT_1<2>	InOut	R6	BANK5	A1 Expansion Connector - 9	
PORT_1<1>	InOut	T8	BANK5	A1 Expansion Connector - 7	
PORT_1<0>	InOut	N7	BANK5	A1 Expansion Connector - 5	
PORT_0<7>	InOut	D1	BANK7		D7
PORT_0<6>	InOut	E1	BANK7		D6
PORT_0<5>	InOut	G2	BANK7		D5
PORT_0<4>	InOut	J1	BANK6		D4
PORT_0<3>	InOut	K1	BANK6		D3
PORT_0<2>	InOut	M2	BANK6		D2
PORT_0<1>	InOut	N2	BANK6		D1
PORT_0<0>	InOut	P2	BANK6		D0
clk	Input	T9	BANK4	50MHz (IC4)	



FPGA Ball Grid Array connections.