DEVELOPMENT OF A DEBUG MODULE FOR A FPGA-BASED MICROCONTROLLER

Rainer Bermbach, Martin Kupfer

University of Applied Sciences Braunschweig/Wolfenbuettel, Germany

Abstract: Description of the development of a debug system for a PIC™-compatible SoC microcontroller implemented in a FPGA. The debug module comprises a hardware interface in VHDL, which communicates with the respective microcontroller components and a software application to present the relevant information to the user and to allow for convenient control of microcontroller functions. Communication between front-end and back-end is accomplished via the JTAG port. The debug module allows easy non-interfering debugging of software for the SoC microcontroller.

Keywords: debugging, microprocessor, hardware, SoC, VHDL, JTAG port.

1. INTRODUCTION

Today, convenient debugging of embedded system software is more or less state of the art. Powerful in-circuit emulators or on-chip hardware assisted debuggers help engineers to develop the programs for all those embedded systems based on standard microcontrollers and microprocessors. Typically, difficulties arise when working with IPs of processors and controllers for building a system-on-chip (SoC). Usually, one has to utilise old-fashioned software debuggers which require at least some system resources by themselves. Interference of the debugger with the target device hinders real-time debugging and often makes full system tests impossible.

At the Computer Engineering Lab of the University of Applied Sciences Braunschweig/Wolfenbuettel a PIC™-compatible microcontroller called VHDL-PIC had been developed and optimised in former projects, see (Bermbach, 2003; Cramm, 2003; Bermbach, 2004; Andreas, 2004). The VHDL-based controller, fully compatible to the cores of the PICmicro™ mid-range MCU family, features typical peripherals such as ports, timer, UART, etc., at a frequency of up to 100 MHz, i.e. with processing power of up to 25 MIPS.

When using the microcontroller in FPGA implementations all the above mentioned difficulties arose. Debugging of system software turned out to be troublesome and tedious. In addition, all software changes had to be run through the software development system. Then the code needed to be transformed into VHDL ROM initialisation code and finally, all the steps of the hardware implementation process had to be performed again. For each error correction all of the above mentioned steps needed to be repeated which was not very effective. To bypass all those problems the idea of an on-chip hardware assisted non-interfering debugger with direct code download was born, see (Kupfer, 2005). The debug module is comprised of a hardware interface (back-end) written in VHDL, which communicates with the respective microcontroller components and a Microsoft Windows™ based software application (front-end) to present the relevant information to the user and to allow for convenient control of microcontroller functions. Communication between front-end and back-end is accomplished via the JTAG port which is available on the FPGA. The debug module allows easy non-interfering real-time debugging of the SoC microcontroller.

The development and implementation of that debugging system is presented in this text. In the

following section a system overview will be given, whereas section 3 describes the hardware interface. Section 4 discusses the implemented debugging functions and section 5 the structure of the back-end/front-end communication. Section 6 gives some detail on the front-end and its implementation and passes over to the conclusion.

2. SYSTEM OVERVIEW

The configuration (see fig. 1) for developing SoC hardware and software with the available VHDL-PIC microcontroller is comprised of a PC running Windows XP™ and a FPGA development board from Digilent Inc. which carries a 200k gate Spartan 3 FPGA from Xilinx. On the PC the free integrated development environment (IDE) MPLAB™ v7.01 from Microchip is used to input source code and assemble it into Intel Hex encoded executable form. Additionally, MPLAB™ may be used to do first software simulations running the built-in simulator.

By means of the graphical user interface (GUI) the user selects the program code to be run on the VHDL-PIC and sends it via the debugging software to the FPGA where it is loaded into internal Block RAM memory configured as ROM. Communication occurs through a parallel port driver serving the JTAG port (IEEE 1149.1) on the FPGA development board. The communication interface inside the FPGA, the so-called TAP controller, handles all read, write and communication requests between the hardware interface and the debugging software.

In addition, VHDL development takes place on the PC utilising Active VHDL™ (Aldec) whereas software from Xilinx (ISE™ 6.3i) synthesises, fits, routes and downloads any hardware code to the Spartan 3 FPGA on the Digilent Spartan-3 Starter Kit Board.

3. HARDWARE

This section describes the underlying architecture and structure of the microcontroller as well as the requirements and the implementation's approach to make the processor "debuggable".

3.1 PIC Microcontroller Architecture

The PICmicro™ controller is a simple but powerful 8-bit processor with a Harvard architecture, a 14-bit instruction word, a hardware stack, a work register which may be involved in nearly every instruction and an internal RAM space of up to 512 bytes organised in four banks, see (Microchip, 1997). The register file (general purpose RAM, GPR) is located in that address space. All special function registers (SFR) of the CPU and peripherals are mapped into this space as well. The controller has single cycle instructions; only jumps, conditional jumps, calls and returns require two cycles. It works with a 2-stage

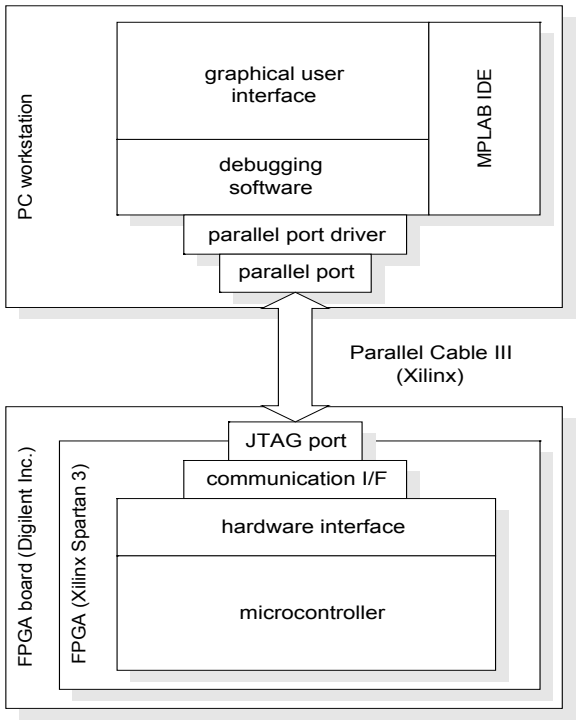

Fig. 1. Block diagram of the development system

pipeline: One instruction is read while the previous is decoded and executed. The machine cycle is divided into four steps called Q1 to Q4. In Q1 the instruction is decoded, in Q2 operands are read, in Q3 the operation is executed and in Q4 results are written back.

3.2 VHDL-PIC Structure

The VHDL-PIC core is fully compatible with the PICmicro™ controller. It not only executes the same instructions as its standard IC counterpart, it was also designed to meet the publicly available timings of the PICmicro™. So the machine cycle stages, interrupt cycles, etc. are fully equivalent.

For an efficient implementation, the instruction ROM uses the Block RAM feature of the Spartan 3 FPGA. Up to two dedicated RAM blocks can be utilised as program ROM giving a maximum code size of 4 Kbytes (2K x 16 bit). The GPR register file is also implemented as Block RAM. The hardware stack may be built from Block RAM or as dedicated flip-flops which is defined in a special library, MyPICPack. The library also determines the size and memory locations of the GPR to adapt the VHDL-PIC to various real PICmicro™ types. In addition, a lot of other configuration data may be manipulated by setting/resetting corresponding constants in the library.

3.3 Debugging Hardware

The functions of the debugger can be divided into hardware and software functions. The means to directly start, stop, read and write, etc. have to be implemented in the hardware interface to the processor core.

First of all, the debugger has to stop the processor to access its internal data. Stopping can only occur at a certain point within the machine cycle (after Q4) so the information gained gives a precise and correct image of the machine's current state. Stopping should be performed upon special request (user break) or upon fulfilling certain qualifying conditions (watchpoints) or arriving at particular addresses in code memory (code breakpoints). Starting the machine resumes processing.

To implement halting a new clock step Q0, located between Q4 and Q1, was created. After finishing the current machine cycle with Q4, the microcontroller normally continues with Q1. Only upon a stop request from the debugger it enters Q0 where the clock is stopped. In this state the controller performs no action and the debug interface has full access to all processor internals. The data and address busses can be used as well as the read and write enable signals. Restarting the machine simply means enabling the clock again and resetting the entering condition for the Q0 state.

The debugger needs read and write access to all the internal registers, memory, SFRs, etc. Stopping the processor grants the use of the internal busses and signals to read and write all of them. Difficulties arise when wanting to read or write the code memory or the stack. Normally no paths to them exist which can be reached via address and data bus. The dual-port character of the used Block RAMs solves the problem, see (Clayton, 2002). Reading and writing through the second port gives access to code memory as well as stack. Using this feature enables code download to the FPGA processor. For ease of implementation also the GPR is implemented as a dual-port RAM.

As mentioned above, the PICmicro™ instruction word is 14 bits wide. When implementing code memory with Block RAM, this RAM is configured as 16 bits wide. The upper two remaining bits can be used to identify different breakpoint conditions which can be written by the debugger. During the run state, hardware checks the two bits to distinguish the various stop conditions. Thus no machine code has to be manipulated to implement breakpoints, etc. and the maximum number of breakpoints is equivalent to the number of code lines.

The above described method of gaining access to the processor resources and data is fully transparent and does not interfere with any normal operation of the VHDL-PIC microcontroller. None of its resources are blocked, no program space, no bytes of RAM nor port pins or interrupts are used. This allows for developing, testing and debugging of the microcontroller software without any interference by the debugger. The only limitation at the moment is that information update can only be performed when the processor is halted. However, the future implementation of trace buffers and dedicated hardware watchpoints will soon allow for real-time information acquisition.

4. IMPLEMENTED DEBUG FUNCTIONS

As mentioned above the major functions created for debugging can be divided into hardware and software solutions. The principle hardware approach was described in the previous section. The currently implemented functions are presented below.

4.1 Hardware Based Functions

The following hardware functions are implemented in the debugging system.

Manual Stop: A manual stop halts the processor to get full access to all memory areas. The new clock step Q0 located between Q4 and Q1 is entered. In this state the PIC™ performs no action and the debug interface has full access. Data and address busses can be used as well as the read and write enable signals.

Stop at Freely Defined Code Breakpoints: A code breakpoint stops the PIC™ if the two upper bits at the current program memory location are set. A signal from the program memory module to the clock module signals this event and the clock process stops after finishing Q4 and entering Q0. Compare to manual stop.

Run: The controller continues with processing after a stop condition by leaving the Q0 state and entering Q1. The formerly read instruction starts to be decoded and a new instruction (for the next cycle) is read into the instruction register.

Single Step: Only one single instruction is performed from stop to stop. The processor leaves the Q0 state starting with Q1 and stops after finishing Q4 and reentering Q0. Single step is implemented with a special signal in the clock module.

Read/Write a Memory Section: When the processor is halted the debugger gains access to all dedicated registers, RAM, SFR, stack and ROM (program memory) by using the PIC™ busses or accessing the second port of the dual-port memories.

Cycle Counter: This feature counts the executed machine cycles by incrementing a 20-bit counter when finishing Q4. The counter can be read or reset.

Reset: This performs a hardware reset by accessing the reset component of the VHDL-PIC.

4.2 Software Based Functions

The following software functions are implemented in the debugging system. They utilise the underlying hardware features.

Set/Remove Breakpoints: Setting and removing code breakpoints are implemented by first reading the program memory at the required position and then setting the bits 14 and 15 for establishing a break-

point or resetting them to delete a breakpoint. To complete the command, the new program word is written back into the program memory.

Run to Cursor: This is used to quickly reach a certain position in the program. It detects the position of the cursor in the code window of the debugger and sets a temporary breakpoint. Then a run is performed. The breakpoint will be cleared automatically once that it has been reached and the processor is stopped.

Step Out of Current Subroutine: This function performs a run and halts the controller on the first instruction after the subroutine call. The hardware stack of the PIC™ is only used for return addresses. Therefore this functionality can be implemented by setting a temporary breakpoint to the current valid return address on the stack.

Step Over the Following Subroutine: When a call is detected, a temporary breakpoint at the position of the incremented program counter (the instruction after the call) is set. If there is no call this is equivalent to a single step.

Fill the Program Memory (ROM): This reads a new program in Intel Hex format and loads the code into the program memory. A reset is performed afterwards and the new program can be executed.

4.3 Special Functions and Solutions

A special situation arises when code breakpoints are placed after jump and call instructions. Due to the instruction pipelining, a new instruction is read while the current instruction is executed. Thus it is possible to detect a breakpoint before the respective instruction is to be executed, which is used in the implemented debugger. When a jump or a call is performed the processor would stop at the breakpoint though the following instruction is not executed. Therefore a special signal is generated if the program counter is not incremented but loaded. The same applies to interrupts. When a breakpoint is not reached due to the execution of an interrupt after the previous instruction the processor stops at the breakpoint after returning from the interrupt.

Some other hardware functions are used especially to control the status of the processor. The “idle” command is used to get the return value of the previous command. The return value of the command “PIC run status” indicates whether the controller has already stopped or is still running. This command is mostly used to detect a stop at a breakpoint.

5. COMMUNICATION AND INTERFACE TO DEBUG HARDWARE

The front-end of the debugger running on a PC needs to communicate with the back-end implemented on the FPGA. Simple debuggers often occupy the UART interface of the microcontroller. This is unacceptable for non-interfering debugging, so another communication path had to be found. As there is already a JTAG port in use to download FPGA programming information this interface was chosen for communication between hardware and software of the debug module. This communication path is defined by IEEE 1149.1 and is also the most common method chosen for many built-in hardware debuggers in standard microprocessors. A Xilinx Parallel Cable III connects to the JTAG port on the Digilent board and to the parallel port of the PC (see fig.1). If running Windows XP™ access to all the peripherals is controlled by the operating system (OS) so a dedicated driver is needed to communicate via the parallel port.

The JTAG port on the FPGA is driven by a TAP (Test Access Port) controller. Apart from other features Xilinx provides access to two user defined shift registers called USER1 and USER2 via the TAP controller. These shift registers are used for the communication interface between the debugging software and the hardware. As the TAP is actually a state machine, a couple of steps, which are hidden in the communication interface software, are necessary to gain access to one of the shift registers. Special software functions have been developed for navigating in the TAP state machine (e.g. for addressing the two user registers and for shifting commands, data or addresses into and out of a user register).

The 8-bit USER1 register is used to transmit short commands such as Stop, Run and Single Step. The 32-bit USER2 register is utilised for reading and writing memory (including the dedicated flip flop registers). Its 32 bits are subdivided into 16 bits for data and 11 bits for address (2K instructions max.). The last five bits select the memory area and differentiate between read and write accesses.

If a response to a transmitted command is necessary the respective register must be shifted again to receive the return value. That means, e.g. for reading memory, first shift in the read command followed by an idle command to receive the requested memory value.

The parallel outputs of the two shift registers are connected to the debug interface (see fig. 2). A signal from the communication interface to the debug interface indicates the reception of a new command. Another signal from the debug interface indicates a return value that has to be transferred into the shift register. Finally this value is shifted out to the PC with the next transmission.

The debug interface consists of two decoders which analyse the output of the shift registers when a new command is received. The decoder for the 32-bit USER2 register is connected to the dual-port RAMs (GPR, ROM, stack) and the busses for reading and writing the memory of the PIC™. Dependent on the command received, the decoder feeds the different busses and signals for reading or writing the requested memory area.

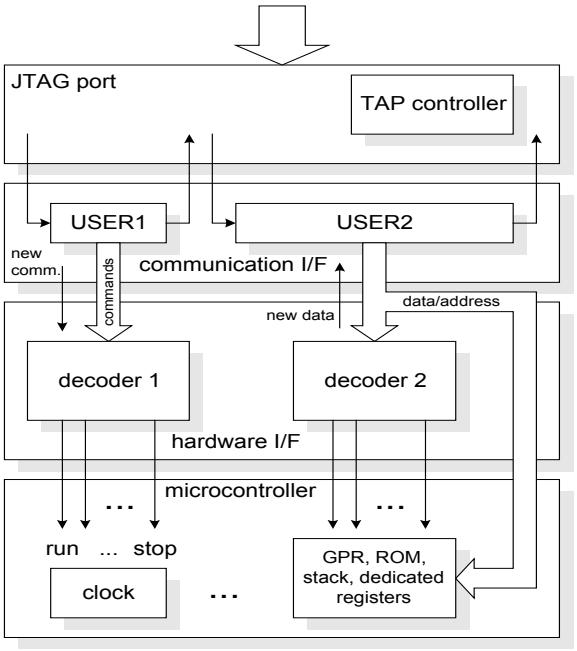

Fig. 2. Block diagram of the debug interface and its connections

The requested value from a read process is stored in the USER2 register and shifted out with the next transfer. In a command, the upper 16 bits contain the address. For security reasons this address is also stored in the lower 16 bits of the return value. The software client can hereby detect errors by comparing the addresses.

The decoder for the 8-bit USER1 register is connected to the components of the processor to initiate commands like run, stop, single step, step out, etc.

To give an example, the communication sequence for a manual stop is first transmitting a “stop” command, then requesting the “PIC run status” and finally, checking the return value with the “idle” command. Once the controller is stopped, a signal marks this state and the debug module has full access to memory and registers. This signal is also evaluated to generate the return value from the “PIC run status” command.

6. FUNCTIONALITY OF THE SOFTWARE FRONT-END

The software front-end is written in C making use of WINAPI functions to implement the graphical user interface (GUI) (see fig. 3). The main window displays the most important information to the user. There are additional windows for displaying the program memory and the GPR (general purpose register) contents. A watch window lists monitored registers and their respective values. Watches can be added or deleted and the value of each supervised register may easily be changed. The most frequently used functions (e.g. run or single step) are placed in the toolbar as icons and can be executed by keyboard shortcuts.

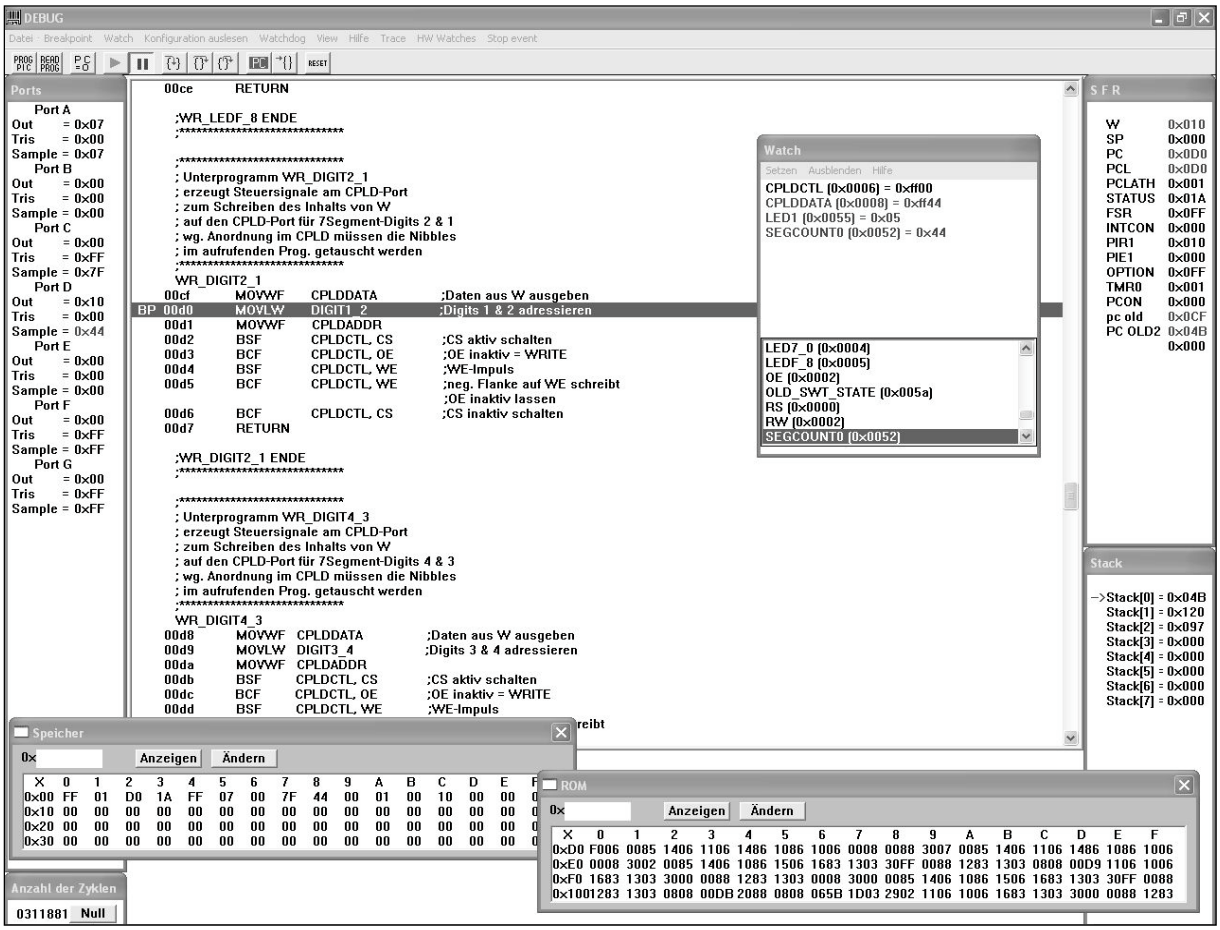

Fig. 3. The front-end software GUI.

In the centre of the application window a code window lists the currently loaded code of the processor. In this window, breakpoints can be set or deleted with a mouse click in the respective line. A “BP” at the beginning of a line marks each breakpoint.

As it is helpful to see the source code of the processor in the code window, a function automatically writes the code into this area by reading the information out of the list file previously created by MPLAB™. This reduces development turn-around time from minutes to seconds. All symbols are read as well and stored in the watch list. This way the user does not have to memorise the address of a symbol but can choose from a list by symbolic name. The current program counter is shown in this window by highlighting the respective line.

Adjacent to the code window all important registers are displayed divided into sections for special function registers (SFR), stack and ports. In the stack section a red arrow marks the current valid stack position. All values of the registers may be changed with the exception of the stack contents.

A software timer checks the status of the microcontroller by transmitting the “PIC run status” command while the processor is running to detect a break. If it is stopped (manual stop or at a breakpoint, etc.) the client software reads all important memory areas and refreshes the main window with the new data. Upon refreshing, new and old data are compared and changes are marked in red.

7. CONCLUSION

The typical difficulties when debugging system-on-chip processors and controllers triggered the development of a powerful hardware-assisted non-interfering debugging module with direct code download.

The debug module consists of a hardware interface and a debug interface both written in VHDL, which communicate with the respective microcontroller components and with the communication interface. The latter supports the transfer of commands and data via the JTAG port to and from a connected PC. It further includes a debugging software and a powerful GUI application running on the PC to present the relevant information to the user and to allow for convenient control of microcontroller functions.

The debug module allows easy non-interfering real-time debugging of software for the SoC microcontroller. It allocates no resources of the microcontroller i.e. no program code, RAM cells, interrupts or communication facilities. It provides typical and advanced functions needed for debugging microprocessor code such as run, stop, single step, breakpoints (max. number equivalent to number of code lines), step over subroutines, step out of subroutines and run to cursor in code window. Naturally, it allows inspection and modification of all registers and memory locations as well as download of new program code avoiding tedious re-synthesis, etc. of the FPGA hardware.

Currently the design allows clock frequencies of up to 80 MHz which can probably be increased in the near future when better qualification of timing paths is realised.

Because of the layered approach, the design should easily be adaptable to other processors. The main adaptations will obviously lie in the hardware interface. Some smaller changes will affect the GUI elements and the debugging software depending on the architecture of the processor.

Currently, developments are underway to enhance the debug module with hardware watchpoints and a trace buffer. With the freely definable watchpoints, one can monitor arbitrary memory locations in GPR or SFR to detect reading from, writing to, or changing of their contents, etc. The trace function will use Block RAMs to store all relevant information (addresses, data, status signals, etc.) of internal events. This will dramatically ease code debugging through real-time data acquisition for later offline inspection which will be especially useful for true real-time processing, for example, with interrupts.

REFERENCES

Andreas, V. (2004). Optimierung eines PIC-kompatiblen VHDL-Mikroprozessorkerns. *Diploma thesis*. University of Applied Sciences Braunschweig/Wolfenbuettel, Germany.

Bermbach, R. (2003). Entwicklung eines Mikroprozessorkerns. *Research report*. University of Applied Sciences Braunschweig/Wolfenbuettel, Germany

Bermbach, R. (2004). Entwicklung eines Mikroprozessorkerns – Bericht über die zweite Phase. *Research report*. University of Applied Sciences Braunschweig/Wolfenbuettel, Germany.

Cramm, I. (2003). Entwicklung eines PIC-kompatiblen Mikrocontrollerkerns in VHDL. *Diploma thesis*. University of Applied Sciences Braunschweig/Wolfenbuettel, Germany.

Kupfer, M. (2005). Entwicklung der Hard- und Software eines Debug-Moduls für einen VHDL-basierten Mikrocontrollerkern. *Diploma thesis*. University of Applied Sciences Braunschweig/ Wolfenbuettel, Germany.

Microchip Technology Inc. (1997), PICmicro Mid-Range MCU Family Reference Manual. Chandler, Arizona, USA.

Clayton, J. (2002). RISC 16f84. http://www.opencores.org/projects.cgi/web/risc16f84/overview.