"TouchScreen" project

Albert Dorn, Eduard Heidt, Jan Helber,
Alexander Irro and Cedric Pilot

System Architecture Document V0.7
written by Alexander Irro

This documentation contains the system architecture document
of the project "TouchScreen TI6"

V0.1	2006-11-5	Initial Release
V0.2	2006-11-6	Bugfixes in ER-Burn (circuit plan)
V0.3	2006-11-11	Deployment and block diagram, updated use cases and fixes
V0.4	2006-11-13	Updated ER-Burn hardware realisation
V0.5	2006-11-17	Control register circuit plan and data flow diagram
V0.6	2006-11-20	Updated ER-Burn activity diagram
V0.7	2006-12-02	CtrlReg flag table, updated diagrams, some small fixes

Powered by LaTeX
generated at 1:38 on December 3, 2006

Contents

1 Preamble

This project is based on the project report of the preceding group from summer term 2006 (SS06) and describes more advanced tasks (debugger, ARM to ER1 program loader, etc.) only. So please refer to their documents to get an idea about the existing system before continuing reading this document.

2 E.R.D.E.

Because of the circumstantial and time consuming development process a completely new idea is born. The E.R.D.E (EinfachstRechner Development Environment) is a tool containing a powerful text editor (ER-Edit) with syntax highlighting and a built-in debugger utility (ER-Debug) featuring step-by-step mode based on ER-Edit to get an improved debug situation. Finally there is a more advanced console based assembler (ER-ASM) and an uploader-tool (ER-Burn) writing the data directly to the ER1-RAM by using the serial interface on a standard PC. On the following figure 1 you can see the E.R.D.E. and its process communication.

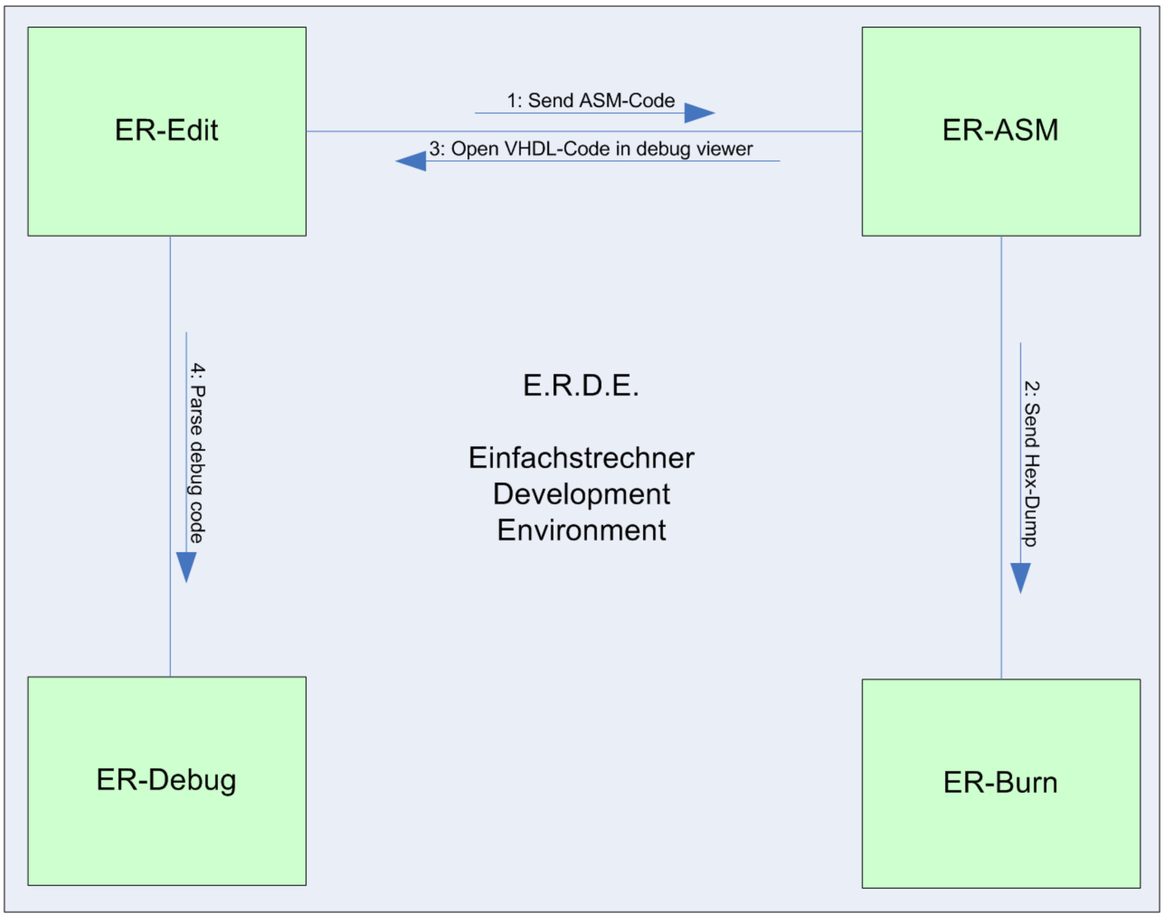

Figure 1: Communication view over all E.R.D.E. parts and their relations.

2.1 ER-Edit - A powerful text editor

The ER-Edit is a complete development suite programmed in Visual C#. It is used to program the ER1 inside the FPGA. It features syntax highlighting and is able to read and write .ER1-Files containing the assembler instructions for the ER1. You are also able to combine this editor with ER-Debug.

2.2 ER-Debug - A Built-In Debugger for ER-Edit

Because of the importance of debugging whilst the development process there is a comfortable debugger. By using a standard serial interface (COM-Port) for communication you are able to parse the assembler code step by step. All necessary registers of the ER1 are displayed besides the assembler code to get a better overview what is happening inside the ER1. By using the debugger the ER1 will be switched into debug mode automatically. This means that the ARM microcontroller will produce the ER1's clock signal instead of the standard quarz oscillator.

2.3 ER-ASM - An advanced console based assembler

This new assembler is based on the java assembler from Markus Honold (SS06) and will be converted into a console application as a common executable file (*.EXE). Such an application can be used much more universal and modular than an all-in-one solution. It converts a .ER1-File containing the ER1 assembler code into a hexdump which can be uploaded to the ER1's ROM directly and also a ready-to-build VHDL source code file for permanent configuration ROM programming using Altera Quartus II. There will be a better feedback to the user telling him more detailed error messages and also the error log can be passed to any other program. According to the customer requirements this assembler implements a comment-feature and symbolic pointers.

2.4 ER-Burn - A data uploader for the ER1

For faster development process there is an uploader tool which burns the data into the ER1's ROM instead of creating a VHDL-File which has to be uploaded to the FPGA manually by using a software like Altera Quartus II. Please note that the ER1-ROM is in reality a RAM so if there is a loss of power the old ER1 script will be restored from the configuration ROM. If you want the new code to remain permanently inside the ER1 you have to burn it into the configuration ROM the old way using VHDL and Altera Quartus II.

2.5 E.R.D.E. Use cases of involed programs

Figure 2 shows all involved programs, their rolls and relations to each other. To give you are more global view we also showed all third party products used in the development process to give successional teams an easier familiarisation to all components of this product.

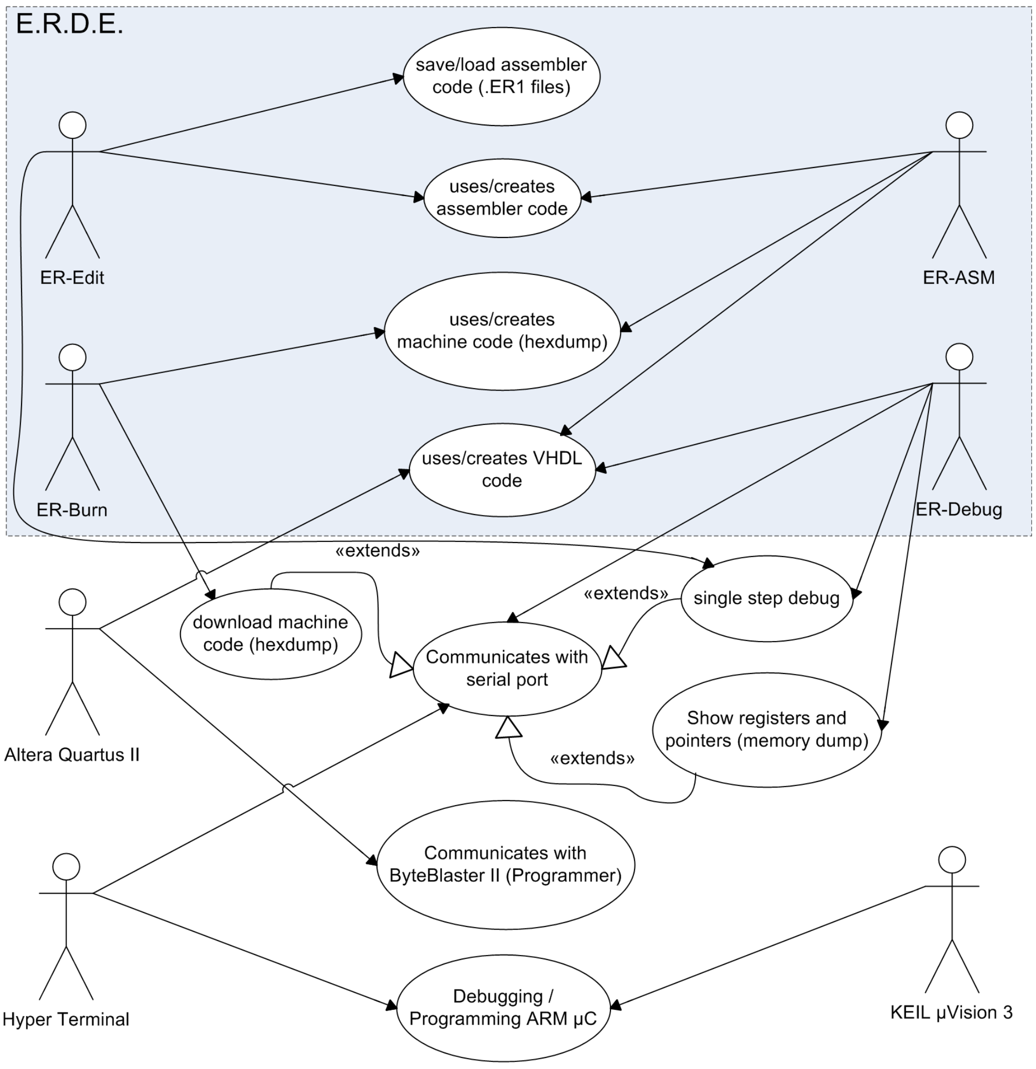
E.R.D.E.
save/load assembler code (.ER1 files)
uses/creates assembler code
uses/creates machine code (hexdump)
uses/creates VHDL code
ER-Edit
ER-ASM
ER-Burn
ER-Debug
«extends»
«extends»
«extends»
single step debug
download machine code (hexdump)
Communicates with serial port
Show registers and pointers (memory dump)
Altera Quartus II
Communicates with ByteBlaster II (Programmer)
Hyper Terminal
Debugging / Programming ARM µC
KEIL µVision 3

Figure 2: Use case of all involved programs and their relations

3 Improved communication for better debugging

We developed a communication interface using a common serial interface available on most PCs. There are defined control signals telling the ARM what to do. For example switching the ER1 to debug-mode by using its external memory control feature via a control register would be at adress 0x83F00020. So there is now a duplex communication between the Host-PC and the ER1 via the ARM: Uploading new programs into ER1's ROM, manual clocking and the other way round reading the ER1's registers and send the data to the Host-PC. By using a common serial interface program like Microsoft Hypertext Terminal you are able to "see" the communication on-screen. All involved hardware devices are shown in the use case diagram in figure 3.

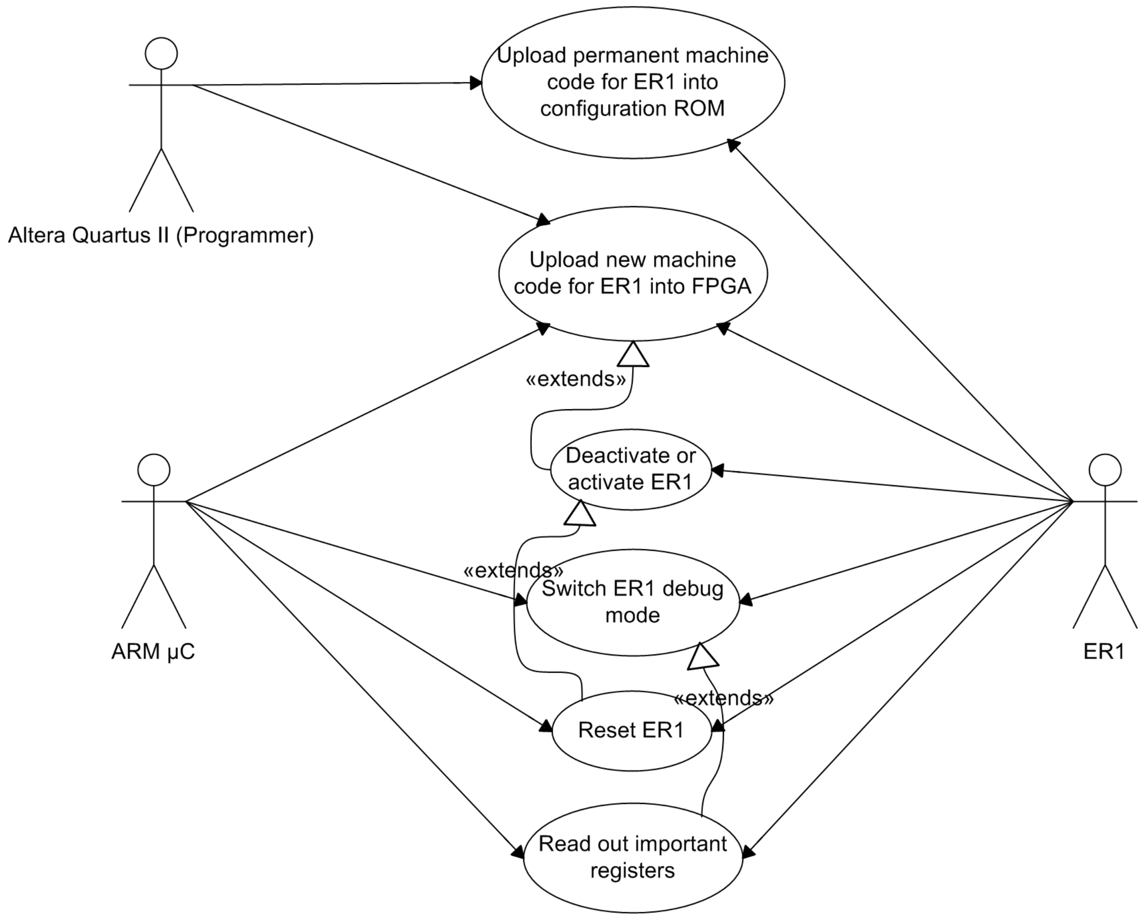
Altera Quartus II (Programmer)
Upload permanent machine code for ER1 into configuration ROM
Upload new machine code for ER1 into FPGA
«extends»
Deactivate or activate ER1
«extends»
Switch ER1 debug mode
«extends»
Reset ER1
Read out important registers
ARM µC
ER1

Figure 3: Use case of all involved hardware devices and their usage

4 Hardware system overview

To provide you a better overview of the system figure 4 shows a deployment diagram with all involved subsystems and their components. To get an idea how all the components are connected there is also a block diagram of important hardware components below.

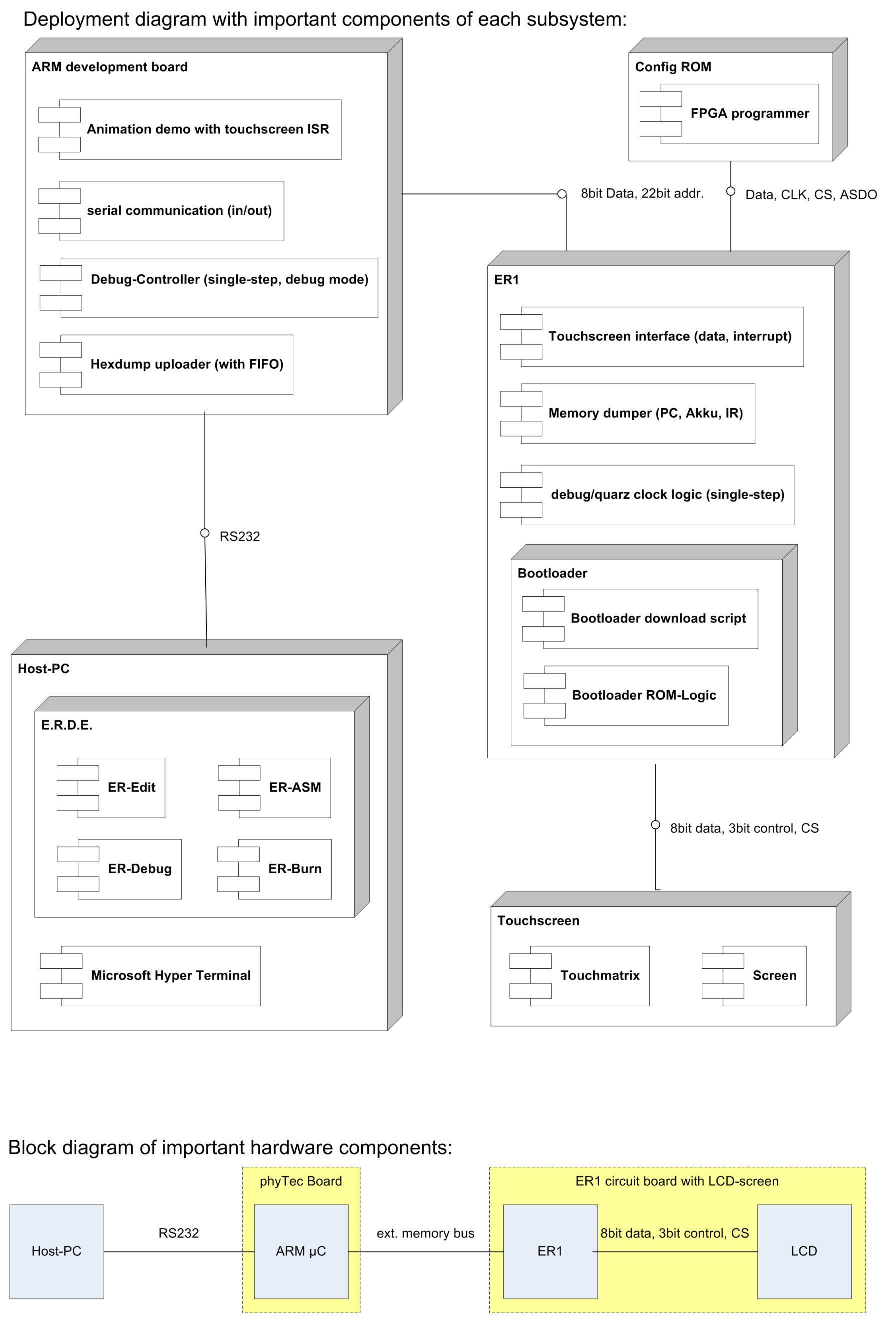
Deployment diagram with important components of each subsystem:
ARM development board
Animation demo with touchscreen ISR
serial communication (in/out)
Debug-Controller (single-step, debug mode)
Hexdump uploader (with FIFO)
Config ROM
FPGA programmer
8bit Data, 22bit addr.
Data, CLK, CS, ASDO
ER1
Touchscreen interface (data, interrupt)
Memory dumper (PC, Akku, IR)
debug/quarz clock logic (single-step)
Bootloader
Bootloader download script
Bootloader ROM-Logic
RS232
Host-PC
E.R.D.E.
ER-Edit
ER-ASM
ER-Debug
ER-Burn
Microsoft Hyper Terminal
8bit data, 3bit control, CS
Touchscreen
Touchmatrix
Screen
Block diagram of important hardware components:
phyTec Board
ER1 circuit board with LCD-screen
Host-PC
RS232
ARM µC
ext. memory bus
ER1
8bit data, 3bit control, CS
LCD

Figure 4: Deployment and block diagram of the used hardware and their connections

5 Hardware debugger implementation

There is a DEBUG_ENABLE flag on address 0x83F00020 inside the ER1 which switches debug modes on and off. If this flag is enabled the clock oscillator of the ER1 is deactivated and the ER1 can be manually clocked with a DEBUG_CLK signal on address 0x83F00024 by setting it to '1'. Because the ER1 has one cycle for loading an instruction and another cycle for executing it, it is necessary to clock the ER1 four times for executing one instruction completly. The three most important registers PC (program counter), ACCU (accumulator) and IR (instruction register) are accessible for the ARM microcontroller at all times at adresses 0x83F00054, 0x83F00058 and 0x83F0005C. Please see the following state chart (figure 5) for more information about the processes:

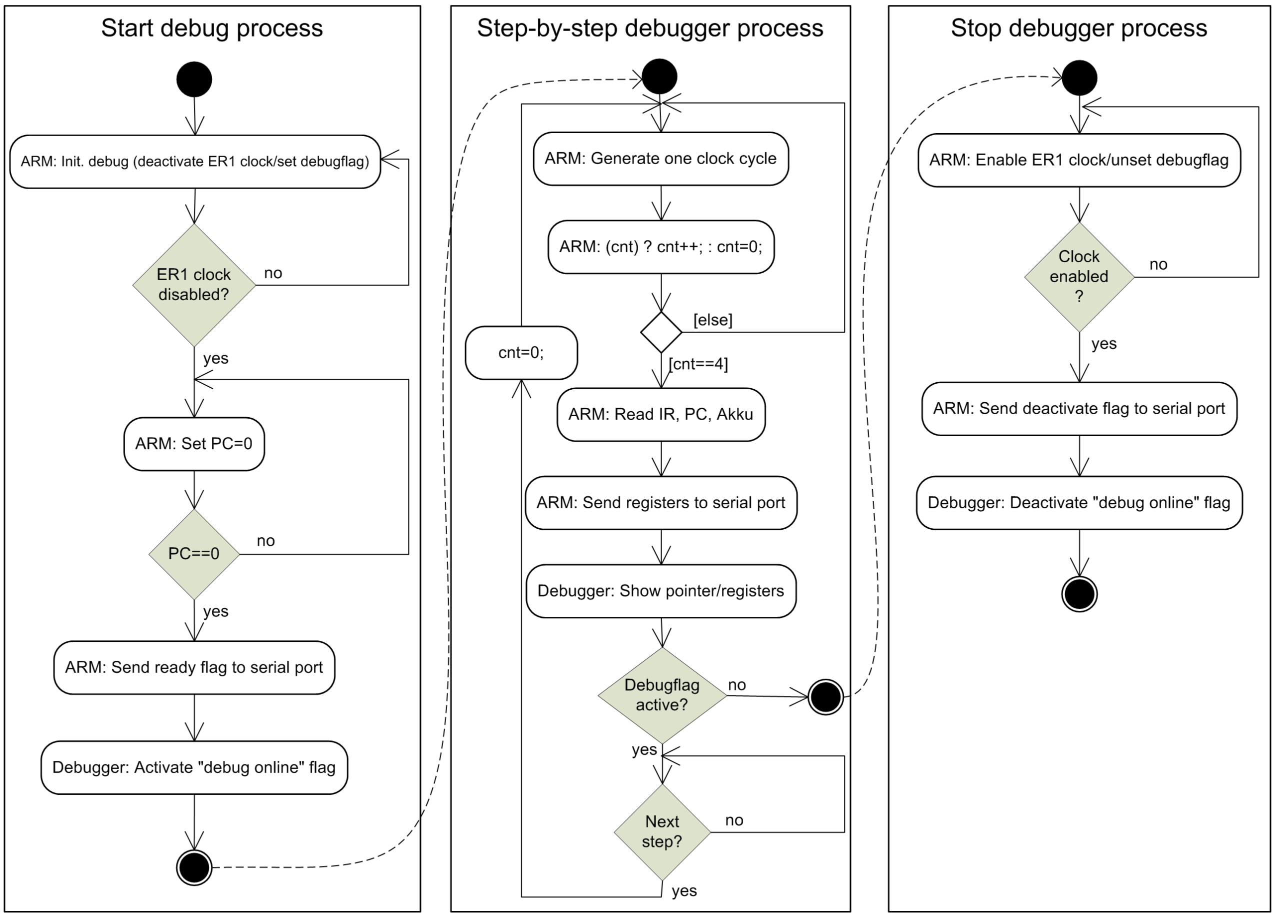
Start debug process
ARM: Init. debug (deactivate ER1 clock/set debugflag)
ER1 clock disabled?
no
yes
ARM: Set PC=0
PC==0
no
yes
ARM: Send ready flag to serial port
Debugger: Activate "debug online" flag
Step-by-step debugger process
ARM: Generate one clock cycle
ARM: (cnt) ? cnt++; : cnt=0;
[else]
[cnt==4]
cnt=0;
ARM: Read IR, PC, Akku
ARM: Send registers to serial port
Debugger: Show pointer/registers
Debugflag active?
no
yes
Next step?
no
yes
Stop debugger process
ARM: Enable ER1 clock/unset debugflag
Clock enabled ?
no
yes
ARM: Send deactivate flag to serial port
Debugger: Deactivate "debug online" flag

Figure 5: Start, stop and step-by-step processes of the implemented debugger

6 ER-Burn in detail

6.1 Principle of ER-Burn

In order to speed up the development process you are able to upload new machine code (ER1 assembler) to the ER1 at any time. All data is transferred via serial interface into the ARM microcontroller and will be buffered in machine code (hex) as a FIFO. Each script has a maximum length of 256x2 Bytes (256 commands) and in case it is shorter than 256 instructions all remaining slots will be set to NOP (no operation). The ARM then activates a bootloader mode by setting the bootloader-flag and resetting the ER1 afterwards. The ER1 will now execute a special bootloader program which overwrites the ER1 instruction ROM with new data out of some transfer registers (explained later). After the transfer the ARM disables the bootloader-flag and resets the ER1 to return to normal operation. In figure 6 you can see the process as an activity diagram instead of a state chart to get an idea about the process timing also.

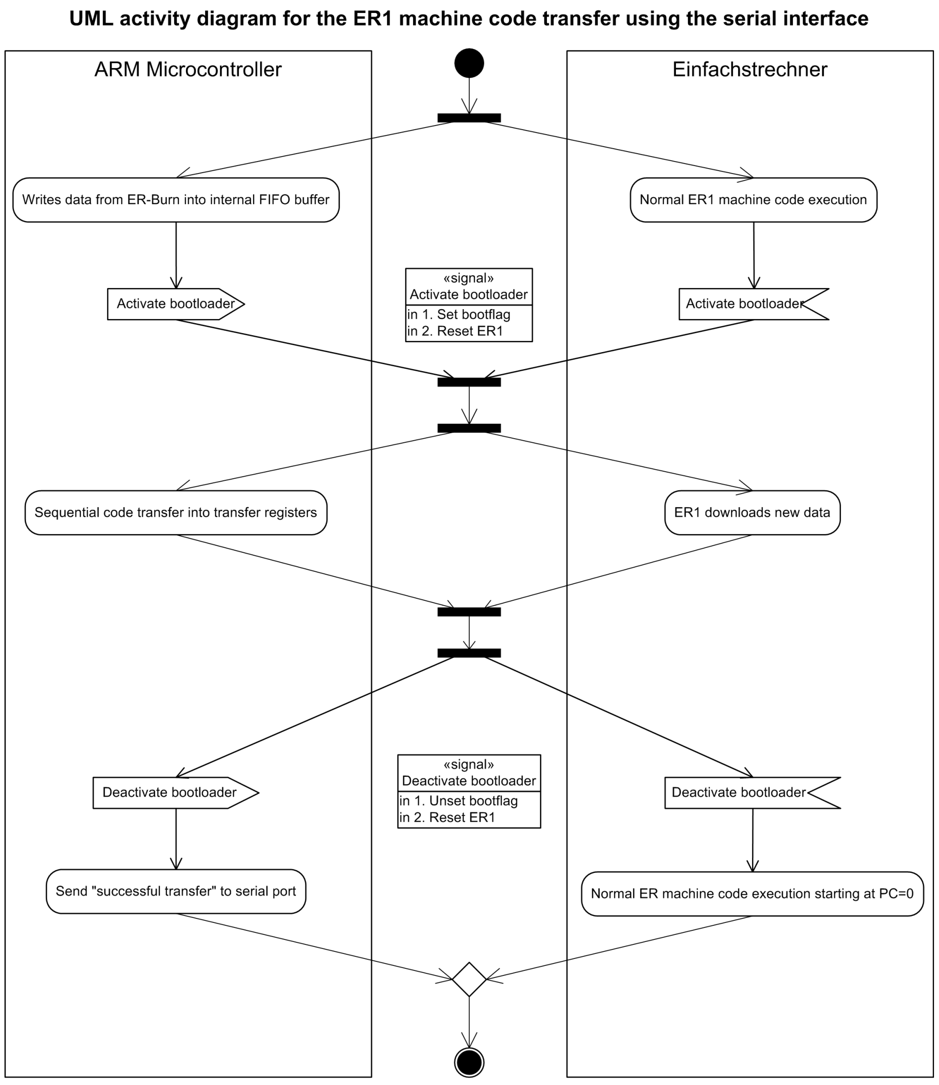
UML activity diagram for the ER1 machine code transfer using the serial interface
ARM Microcontroller
Einfachstrechner
Writes data from ER-Burn into internal FIFO buffer
Normal ER1 machine code execution
«signal»
Activate bootloader
in 1. Set bootflag
in 2. Reset ER1
Activate bootloader
Activate bootloader
Sequential code transfer into transfer registers
ER1 downloads new data
«signal»
Deactivate bootloader
in 1. Unset bootflag
in 2. Reset ER1
Deactivate bootloader
Deactivate bootloader
Send "successful transfer" to serial port
Normal ER machine code execution starting at PC=0

Figure 6: State view of both ARM and ER1 activities while uploading new machine code

6.2 Hardware realisation as a bootloader

We changed our first idea of a two way accessible ROM to a bootloader based idea involving the ER1-CPU into the download process. Later we will have the possibility to change the source of data like USB, parallel port or another microcontroller. An ER instruction has 16 bit, but there is only an 8 bit data bus connecting the ARM so two separate 8 bit registers are necessary named DatHigh and DatLow. For managing the transfer there is also an 8 bit control register with a bootloader flag telling the ER1 to switch into bootloader-mode after the ARM triggered a reset of ER1. By doing so, the ER1 will load its intructions from a bootloader ROM containing an assembler program that copies new data into the normal ER1-ROM/RAM. Other flags in the control register are described in the table at the end of this page. After the transfer has finished the ER1 is resetted again and returns to normal execution of the loaded program. The ER1-ROM is only writable in bootloader mode and will behave like a ROM in normal operation mode.

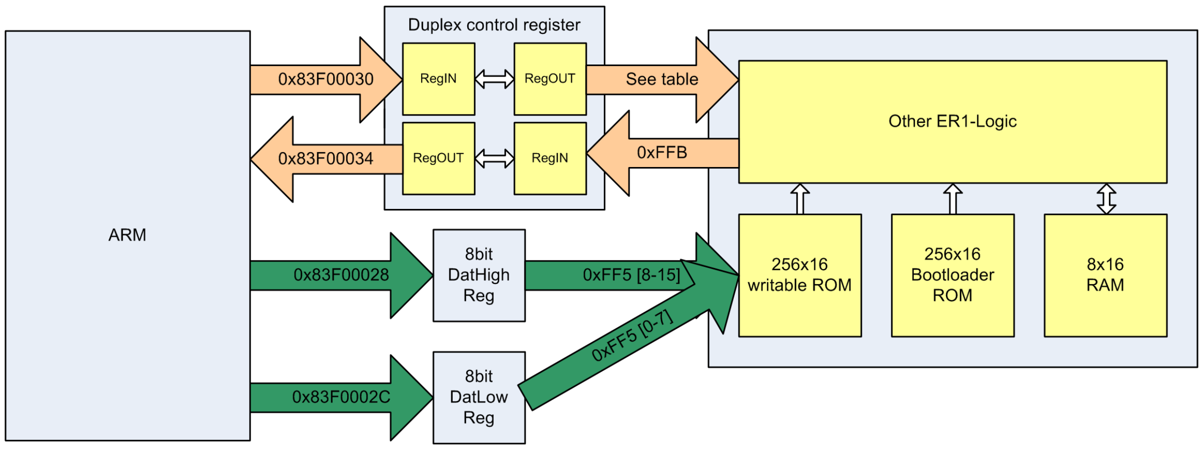

Figure 7: Block diagram of bootloader data flow between a source (ARM) and ER1

The following table describes all flags of the 8 bit control register accessible on address 0x83F00030 (ARM to ER1) and 0x83F00034 (ER1 to ARM). Until now only the lowest four bits are assigned. The highest bit (bit 8) is CtrlRead and is read only.

Flag	**ER1-Adr**	**Description**
Bootloader	0xFF8	This flag tells the ER1 on startup after a reset to switch into bootloader mode
NewCmd	0xFF9	If the ARM stored a new command into DatHigh/DatLow this flag is set. As soon as the ER1 has read the new data the flag is set to 0.
Debug	0xFF6	Enables step-by-step mode by disabling the ER1 clock and using manual clocking This flag is prepared but we still use a data flipflop at adrress 0x83F00020.
DebugCycle	0xFF7	Manual clock flag for triggering one cycle. This feature is reserved for future developments. At the moment we just use the rising edge of the address signal (0x83F00024) inside the clocking unit of the ER1 for manual clocking.
CtrlRead	0xFFA	This is a flag controlled by the register itself. It is 0 if new data is in the control register and will be 1 as soon as a chip select occurs to show both CPUs if they read the newest flags out of the control register (not DatHigh/DatLow!).

6.2.1 Circuit plan of the control register

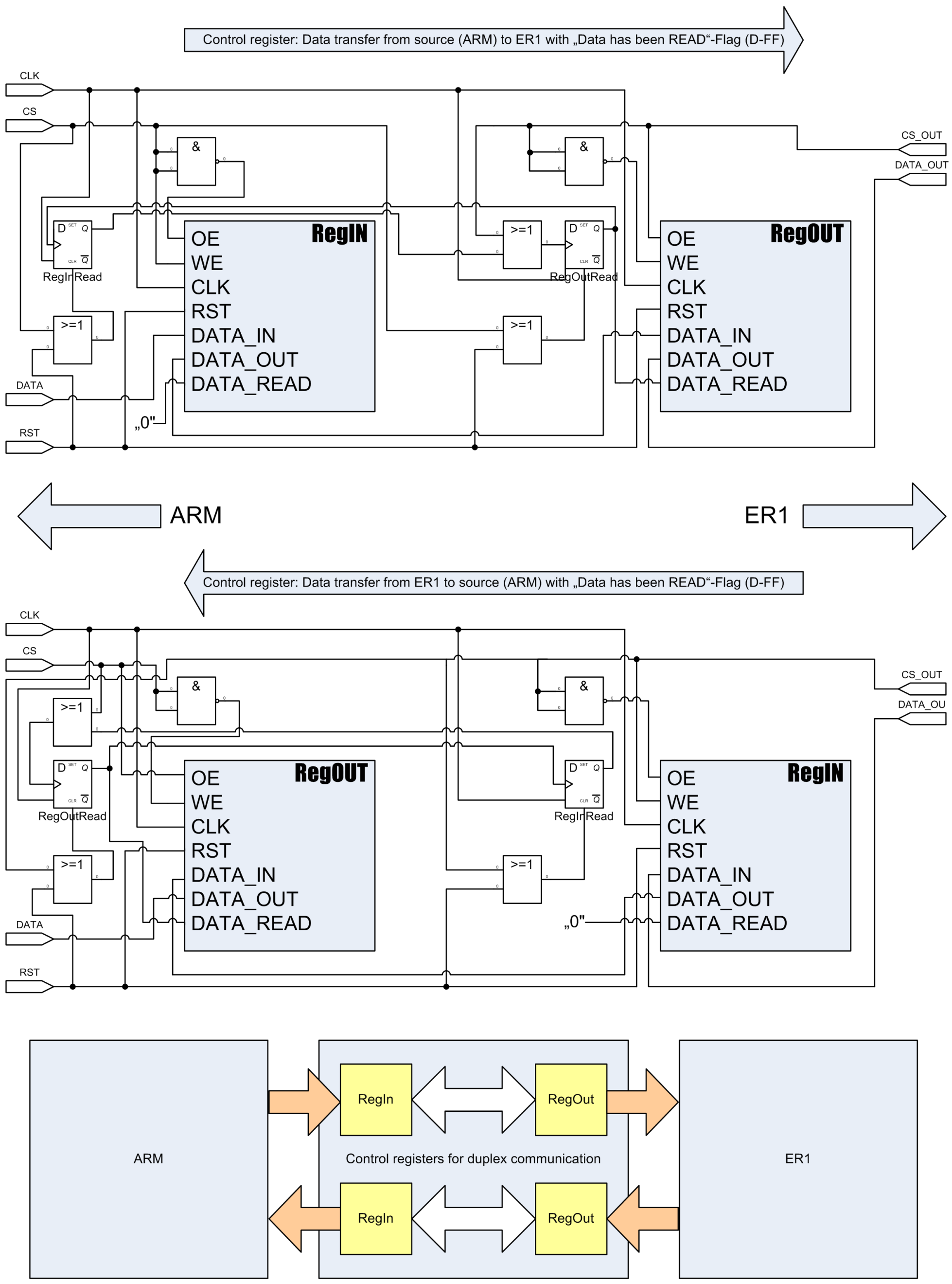
Control register: Data transfer from source (ARM) to ER1 with „Data has been READ"-Flag (D-FF)
CLK
CS
DATA
RST
CS_OUT
DATA_OUT
&
>=1
RegInRead
RegOutRead
RegIN
RegOUT
OE
WE
CLK
RST
DATA_IN
DATA_OUT
DATA_READ
„0"
ARM
ER1
Control register: Data transfer from ER1 to source (ARM) with „Data has been READ"-Flag (D-FF)
DATA_OU
RegIn
RegOut
Control registers for duplex communication

Figure 8: Detailed view of the control register managing duplex communication

This duplex register is built out of four 8 bit registers. As seen on the bottom of figure 8 the upper two are used for ARM to ER1 communication and the other two registers vice versa. Each pair is chained so RegIN can store data at any time but RegOUT will inherit the new data only with a clock cycle and an inactive output chipselect from RegIN for decoupling reasons. As soon as an output chipselect occurs the DATA_READ-Flag is set which is stored in the chained data flipflops RegInRead/RegOutRead. If new data is stored the DATA_READ-Flag is set to 0 by resetting the two data flip flops.

7 Used software packages

7.1 Preceding software basis

This project is based on several preceding and open source software projects.

7.1.1 Scintilla

It is a free source code editing component for Microsoft Windows featuring text highlighting. It is published under the GPL and is available at http://www.scintilla.org.

7.1.2 ARM animation library

There is an already working animation demo showing symbols, graphics and text on the LCD screen. This library was combined with our new debugging unit and is used to show some responses of the touchscreen.

7.1.3 ER1 touchscreen polling

The group of the summer term had the idea to use the so called "Einfachstrechner" as a controller for the LCD touchscreen. It polls the touchmatrix periodically and sends an interrupt to the ARM board if a keypress occured. This solution still had a lot of bugs and so we build up our debugger based on this project to detect all pending bugs. Most parts of this project are programmed in VHDL and Java.

7.2 Used Tools

7.2.1 Altera Quartus II

The FPGA and configuration ROM devices on the LCD board are featuring a JTAG interface. We use it to program/burn the ER1 architecture and the associated ER1 script. There is a Web Edition of Quartus II which is freely available on Altera's homepage.

7.2.2 Keil uVision 3

This is KEIL's standard software tool for programming and debugging the ARM7 microprocessor based boards like our Philips LPC2294 chip. In combination with the KEIL uLink USB device you also get a powerful debugger with breakpoints and single step mode showing and simulating all registers and associated on-board hardware (serial ports, oscillator, etc.). KEIL uVision 3 has a basic version which is free of charge and meets our requirements completely.

7.2.3 Eclipse for Java

The group of the summer term developed their assembler in Java. We altered it to a command line tool and therefore it was necessary to implement the new features and changes. Eclipse is available as open source.

7.2.4 Microsoft Visual C# 2003 Express

A powerful development environment is necessary for our E.R.D.E. because we need a good GUI for maximum comfort. Fortunately Microsoft is giving out a limited express version for free.

7.2.5 Microsoft Visio 2003

This software has an easy to use diagram editor featuring most of the UML diagrams and a lot of other shapes and figures that can be exported into PNG or JPEG files.

7.2.6 $\LaTeX$

For the documentation we use $\LaTeX$ exclusively because it is fast, easy, supports Adobe PDF output and finally is open source software.

7.2.7 SubVersion

We are a group of five developers that work parallel on most parts of the E.R.D.E. project. To avoid overwriting of one's source code or documentation we use the open source server SubVersion (SVN).