PBCC:
PicoBlaze C Compiler
User Manual

Zbyněk Křivka

krivka@fit.vutbr.cz

Report for
Virtual Laboratory of Microprocessor Technology Application
2C06008 supported by MŠMT

© 2008-2009, Zbyněk Křivka
http://www.vlam.cz
UNIS spol. s r.o, R&D; ÚTIA CAS ČR; VUT Brno, FIT, DIFS

Content

© 2008-2009, Zbyněk Křivka
http://www.vlam.cz

Revisions

No.	Date	Author	Revision description
1	2009-12-30	Z. Křivka	First draft

Abstract

PicoBlaze-3 is 8-bit soft-core processor developed by Xilinx Company used for simple state control application in IP cores in FPGAs. This document describes the development of C compiler. In details, the implementation covers only basic aspects of language C because of the complexity that cannot be handled in such small processor. To deal with the standard syntax and some semantics of C, we use SDCC framework focused on 8-bit CPUs. As the user manual, this document describes the command line interface of the compiler (abbrev. PBCC). For more details about the architecture and implementation of PBCC, see PicoBlaze C Compiler report.

Keywords

Compiler, parser, language C (ANSI-C), 8-bit processor, PicoBlaze, KCPSM3, SDCC, Squeak Smalltalk, Front-end, Back-end

Terminology

JSON format = easy to use text-based format to load and store data in structures (heterogeneous) and lists (homogeneous); Suitable for interlanguage/interplatform structural data in programming languages such as JavaScript, C/C++, Smalltalk, Python, etc. (see [4])

IR = intermediate/internal representation (e.g. 3-address code)

IS = Instruction-set

PBCC = PicoBlaze C Compiler

SDCC = Small Device C Compiler

© 2008-2009, Zbyněk Křivka

Introduction

Xilinx PicoBlaze version 3 (for FPGA Spartan 3, [1]) is very tiny and simple soft-core processor by Xilinx. It provides one interrupt, small amount of memory, and the limited instruction set with the basic arithmetical-logic instruction and branching. This processor can be programmed in the specific assembler defined by Xilinx. More specifically, there are two dialects of the assembler for version 3. First, official assembler used by VHDL and HEX generator (KCPSM3), and the second, assembler accepted by pBlazeIDE graphical simulator of this processor (see PicoBlaze manual [2]).

Architecture

As there are several compiler frameworks such as GCC, clang for LLVM, and SUIF2, we use existing C front-end that handles target-independent optimizations and focuses on back-end design, implementation, and optimization. We have chosen to extend Small Device C Compiler (SDCC).

The high-level design of PBCC is shown in Figure 1. The SDCC front-end starts with SDCPP (adopted GCC preprocessor) and GNU Bison to make syntax-directed translation of C source into IR (list of iCodes consisting of 3-address code with additional high-level information such as type information).

As there is no solid interface for a back-end implementation in SDCC and the documentation of this part is totally insufficient, we introduce a new interface between SDCC front-end and PBCC back-end that is implemented by export and import of JSON file format of IR. Furthermore, to enable rapid prototyping, better abstraction support, and debugging comfort, we develop PBCC back-end from scratch in Pharo Smalltalk environment (see [6, 10]). The back-end separation and from-scratch development is also supported by extremely low memory resources and non-standard minimalistic target IS architecture that misses native data stack and some arithmetic instructions such as multiplication that must be efficiently simulated. In the end, PicoBlaze C Compiler is composed of several chained command-line tools (see [8, 9]).

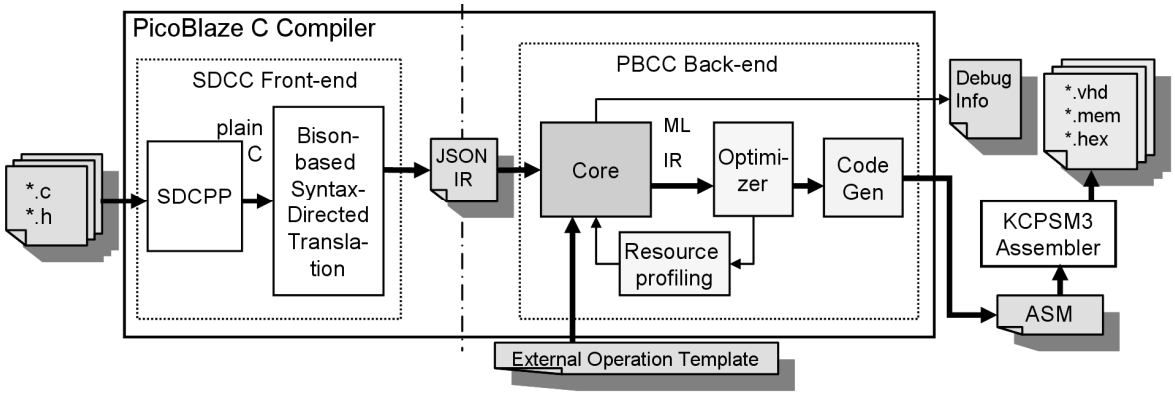

Figure 1. PicoBlaze C Compiler (PBCC) Architecture composing of front-end and back-end connected by IR in JSON format. For the classical list of compilation phases, see [4].

Summary of additional compiler features:

- control flow graph analysis – the absence of indirect function calls and small amount of memory allow the generation of full control flow graph and using global optimization algorithms;

© 2008-2009, Zbyněk Křivka

- due missing native data stack, it is simulated and used for the storing of activation records (stack frames) for static function calls;
- due to the very low memory capacity, only the most essential global types, macros, variables, and functions will be implemented as the torso of standard C library. With so small architecture, modularity is a useless luxury, so PBCC supports source-code level inclusion only (no modularity);
- as there is no application binary interface by the manufacturer and no linkable code is generated, linker tool is not necessary;

SDCC Front-end

SDCC framework is an open-source C cross-compiler (written in C) developed primarily for 8-bit MCU with several back-end implementations for different MCUs (Microchip PIC, Intel 8051, Motorola HC08, etc.). In comparison with GCC and SUIF2, SDCC uses low-level 3-address intermediate representation (IR), called iCodes. Moreover, for every investigated framework, its IR assumes the support of a native data stack.

The standard C language (from 1999) is augmented by some embedded system specific properties and syntax extensions such as embedded assembler code, interrupt handler declaration, or the specification of variable location and address. Some features are added by pragma (optimization control).

For more details about SDCC framework, see report SDCC Framework (in Czech) or official SDCC documentation ([7]).

PBCC Back-end

The back-end internal architecture is based on iterative transformation of multi-level IR (ML IR) towards the optimized target code representation based on resource profiling information to handle essential requirements on the memory unassumingness. The implementation of IR and transformations uses the combination of interpreter and visitor design pattern.

Further, non-native operations (e.g. multiplication) are either (1) outsourced by external peripherals, or (2) simulated by PicoBlaze IS (quit memory consuming). Today's implementation supports the first one only.

The resulting assembler code (ASM) is together with VHDL/Verilog templates compiled by KCPSM3 Assembler into the resulting hardware description of PicoBlaze containing the program hard-coded in its memory. Finally, the hardware description can be interconnected with the rest of the embedded system such as external operations implementation, synthesized to a bit-stream, and downloaded into an FPGA chip.

PBCC Usage

Primarily, PBCC is design as a command-line tool that can participate as a component of a bigger compilation system. Included batch files support Windows platform at the moment. We plan also to support the Unix-based platforms.

Compiling System

The tool chain provides the complete compilation service from C source code to the PicoBlaze-3 assembler that can be simulated or to the HEX file that can be downloaded into FPGA.

1) The compilation of C source code into PicoBlaze-3 Assembler (by PBCC)

a. SDCC-based Front-end (tool `sdcc` and its command-line parameters are described in the SDCC manual or SDCC Framework report) produces 3-address code (iCode instructions) in JSON format;
b. PBCC Back-end (implemented in `Pharo` Smalltalk that is quiet non-typical environment to be used as a command-line tool) processes JSON format of iCode instructions and generates the target code (PicoBlaze-3 Assembler).

2) The compilation from PicoBlaze-3 Assembler to HDL description (VHDL or Verilog) of the processor with the corresponding memory content image (`KCPSM3`)

Now, the user can integrate the processor into a new FPGA design (using Xilinx Design Tools such as Xilinx ISE or Xilinx EDK) or just use it as a new processor code for existing one (using for example QDevKit for FITKit demonstration board).

Command-line parameters

This section describes the command-line parameters of the tools in the PBCC Compilation System. First, we describe the compilation system based on a batch file `pbcc.bat`.

Second, we describe PBCC specific parameters of SDCC front-end (Phase 1a). In addition, we repeat the most useful arguments used by SDCC. Third, we introduce all supported and planed arguments of PBCC back-end (1b) using Pharo Smalltalk Virtual Machine.

In the end, we give some tips about the simulation of the resulting PicoBlaze-3 Assembler code and creating FPGA bitstream with the PicoBlaze-3 processor using integrated development tools such as Xilinx ISE or QDevKit. As the PicoBlaze processor is developed by Xilinx company, we are focused on the FPGAs from Xilinx too.

The main compilation batch file is `pbcc.bat` with the following usage:

```
pbcc.bat source.c [kcpsm3|pblazeide] [pause]
```

The first parameter is the filename of the source code (**`".c"` extension is mandatory**) to be compiled. Next parameter chooses the dialect (kcpsm3 is default) to generate. The last parameter (`pause`) enables pauses between each phase of the compilation system (for debugging purposes; implicitly no pauses). To ensure valid paths linking to the required tools modify four environment variables in the batch file (paths are relative with respect to the current directory of the batch file):

```
@SET SDCC_PATH="..\sdcc\bin_vc"          ; directory of sdcc.exe and sdcpp.exe
@SET PBCC_PATH="..\pbcc"                 ; directory of Pharo VM and its image
@SET ASM_PATH="..\KCPSM3\Assembler"      ; directory of KCPSM3.exe
@SET SIMUL_PATH="..\pbcc"                ; directory of pBlazeIDE (pBlazIDE.exe)
```

The result of the compilation chain executed by `pbcc.bat` is the HEX file of the same name as the input C file (e.g. `source.hex`). There are created also some temporary files useful for the debugging (`%PBCC_PATH%\source.json` and `%ASM_PATH%\source.psm` if used `kcpsm3` argument).

PBCC Front-end Command-line Parameters

The list of SDCC arguments is augmented by several new one to support new PicoBlaze port (or JSON IR export if you want).

`--json`
enables the output of iCode instructions in JSON format to be processed outside the SDCC

`--json-file="filename.json"`
sets the filename used to save the exported iCode instructions in JSON format

PBCC Back-end (Pharo) Command-line Parameters

Next, we list the complete list of supported parameters of PBCC back-end to generate target code in PBCC compilation system. The arguments are case-insensitive. They are passed to Pharo Smalltalk Virtual Machine without checking or changing. The syntax of the arguments has to be satisfied carefully; otherwise unexpected behavior can appear.

`--input=input.json`
specifies the input filename of the imported iCode instructions in JSON format that are generated by PBCC front-end (Default: funstructDump.txt)

`--output=output.asm`
sets the output filename with the target code (PicoBlaze-3 Assembler) (Default: output.asm)

`--errorLog=error.log`
sets the filename of log to notify the compilation and Virtual Machine errors (Default: error.log)

`--warningLog=warning.log`
sets the filename of log to notify compilation warnings (Default: warning.log)

`--log=log.log`
sets the filename of log for statistics and additional information (Default: log.log)

`--dontClose`
means that Pharo Smalltalk environment will not be closed after the compilation process executed from the command line. This is useful for back-end debugging and development. (Default: false)

`--dontProcess`
means that Pharo Smalltalk environment is only opened and no action according to its arguments is done. This is useful for back-end debugging and development. (Default: false)

`--target=PicoBlaze3`
chooses the target platform. Now, only `PicoBlaze3` processor is supported by the PBCC back-end. For other target platforms, see SDCC manual. (Default: `PicoBlaze3`)

`--dialect= kcpsm3|pblazeide`
selects the assembler dialect for the chosen target platform (see argument `--target`) as there are some minor differences between the PicoBlaze-3 Assembler for HDL/HEX production (KCPSM3) and for the simulation (pBlazeIDE). (Default: `pblazeide`)

Future plans (Command-line Parameters)

NOT SUPPORTED: `--version`

The log file contains the version information (version, revision, date) after the translation.

NOT SUPPORTED: `--debug`
Adds the annotation to the output target code (e.g. in DWARF format, probably in an extern file) to support the source code level debugging of C language. (Default: false)

NOT SUPPORTED: `--verbose`
generates even more detailed reports into log files including detailed comments directly into the resulting assembler with the emphasis on the human readability. (Default not specified yet)

NOT SUPPORTED: `--inline-dialect= kcpsm3|pblazeide`
selects the assembler dialect for the inline assembler in C source code (can be different from `--dialect` argument) (Default: `pblazeide`)

NOT SUPPORTED: `--simulate=program.asm`
executes the command-line simulation of the specified assembler file. This argument has to be augmented by the configuration of the simulation (see argument `--simulationConfiguration`)

NOT SUPPORTED: `--simulationConfiguration=simulation.cfg`
passes all necessary configuration (in XML or INI format) to the command-line simulation such as the program file, starting code line, processor initial state, memory initial content, inputs, storage for outputs, etc.

NOT SUPPORTED: `--opt=code|memory`
chooses the target-dependent optimization to be focused on the size of the instruction code (`code`) or the amount of memory (`memory`). (Default: no such optimizations yet)

NOT SUPPORTED: `--stackSize=0-64|dynamic`
sets the maximal size of the simulated data stack. The dynamic approach means that it is limited by hardware conditions. (Default: `dynamic`)

NOT SUPPORTED: `--ISR-function-number=::DIGIT::`
specifies the number of the interrupts in the interrupt table/vector (ISR table). (Default: 1)

Example

Download the 32-bit Windows binary package from the webpage of VLAM project. We assume that you preserve the directory structure in the package and you decompress it in the root of system disc (C:).

1a) SDCC compiler execution (for more inspiration, see `pbcc.bat`):

```
C:\pbcc\sdcc\bin_vc\sdcc.exe -c –IC:\pbcc\sdcc\device\include\picoBlaze
-mpicoBlaze --all-callee-saves --json --json-file="<json-output-file>"
<program.c>
```

1b) PBCC Back-end execution:

```
C:\pbcc\pharo\Pharo.exe pbcc.image --input=<json-output-file> --output=<asm-output-file.asm> --errorLog=error.log
```

2) After the execution of the back-end you maybe notice a blink of the graphical user interface of Pharo Smalltalk. You should ignore it (or see `--dontClose` argument of the back-end).

Now, you can simulate the PicoBlaze-3 Assembler in **pBlazeIDE** (see section Simulation Tips) or continue the translation process by **KCPSM3** to get VHDL and HEX files useful to generate bitstream for FPGA (see section Binary Image and VHDL).

Simulation Tips

pBlazeIDE (pBlazIDE.exe, see [3]) is a simulation environment with the graphical user interface for Microsoft Windows. It can simulate several different assembler dialects such as Picoblaze I, II, 3, and CR.

To simulate the dialect *pblazeide* produced by PBCC, execute the GUI of the simulator, check menu *Settings* that item *PicoBlaze* 3 is selected. Now, we can import the generated assembler file (**.psm*) by menu *File→Import* to satisfy the specific syntax requirements of pBlazeIDE. **Warning:** Do not use menu *File→Open* until the file is saved from pBlazeIDE. To initialize the simulation process and escape editing mode, use menu *Simulation→Simulation*. The simulation toolbar provides basic stepping features for simulation and debugging such as reset, step one, step over, breakpoints, etc (see menu *Simulation*).

Binary Image and VHDL

To use Xilinx PicoBlaze-3 processor and the program written in its assembler (PicoBlaze-3, `*.psm`) you have to generate binary image of the instruction memory and HDL (Hardware Description Language) templates that can be integrated in your hardware design.

KCPSM3.exe (see [1, 2]) is a command-line tool processing the PicoBlaze-3 Assembler and producing the HEX and HDL (VHDL or Verilog) files. The only argument is the filename with the assembler. Next, files `ROM_form.{vhd|v|coe}` are required. It generates several resulting files (approx. 15; for more details, see PicoBlaze manual) including ROM content and some statistics (the list of labels, the list of constants, etc.).

PBCC Compilation from Source Code

As there is new major version of SDCC (2.9.0), we are porting the whole front-end project to this version. Then, we will introduce step-by-step tutorial to compile the SDCC source codes including PicoBlaze-3 port in Visual Studio 2008 C++ Express (Free Edition).

The Back-end project in Monticello is private at the moment. The minimalistic image is based on the PharoKernel [11] project to save space and computation time during the production compilation. This image does not support execution with graphical user interface and Pharo debugging environment.

Nevertheless, in the future public development version of the image of Pharo, the source code can be read directly from the image. Next version of the back-end will be probably made public in Squeaksource.com (Web-based version control system for Squeak/Pharo Smalltalk environments).

© 2008-2009, Zbyněk Křivka

For example, you can open Pharo development image in the development mode by `--dontProcess` argument. Then, you can explore the code in System Browser or debug the execution of some code in Workspace:

```
PbccLauncher new startUpWith: (Dictionary newFrom:
      {'--DONTCLOSE' -> true.
       '--DIALECT'->'kcpsm3'.
       '--INPUT'->'demo.json'}) handleErrors: false.
```

Source code policy

As the compiler is still in the rapid development, we do not provide the source code of the modification of SDCC (GPL license) yet, but we will do that before the end of the project (summer 2011). On the other hand, back-end part of the compiler is contained in Pharo image. The binary package includes the Pharo Virtual Machine (MIT license) and the image with back-end code.

Specific Syntax and Semantic of PBCC

Since SDCC works as a PBCC's front-end, it analyses standard C99 language (see command-line arguments to modify the version of the standard processed by SDCC). We shortly revise essential changes and updates in SDCC with respect to the C99. We focus on explaining PicoBlaze-specific syntax and semantics modifications with respect to SDCC.

C99 Extensions in SDCC Front-End

To accomplish specific requirements of embedded systems, SDCC introduces several new keywords to handle hardware specific problems in high-level C language (see [7]).

`__reentrant` (NOT SUPPORTED YET!)
This keyword placed behind the function declaration denotes a function that can be called by other functions. That is, we have to deal with auto/local variables and recursion properly including the standard work with the data stack. By default, a function is not reentrant.

`interrupt (?1) using (?2)`
This pair of keywords denotes the function that works as an interrupt handler. PicoBlaze processor supports only one interrupt, that is, `(?1)` parameter is always constant (We plan the simulation of interrupt vector in the future). `using` parameter denotes used registers (bank number) in the interrupt handler.

Example:
```
void interrupt_handler(void) interrupt
             { /* the handler definition */ }
```

`__asm … __endasm;`
The pair of keywords encloses the inline assembler code (processor-specific). SDCC makes no validation of this code. PBCC checks the syntax and the registers usage (including the references to local variables by "_ID" notation).

`__data __at (?1)` (NOT SUPPORTED YET!)
This keyword denotes the absolute address (`?1`) of the memory location of the program entity such as variable, literal, and function.

Warnings and Error Messages

Every tool in the compilation system has its own messaging system to inform the user about the problems, warning, and statistics.

SDCC and KCPSM3 as third party tools just warnings and error into standard error output and specified logging files (if required).

The back-end stores all statistics (the number of generated instructions) into log file (`.log`), warnings (the insufficient memory problem) into warning log file (`.wrn`), and errors (input cannot be opened, internal errors) into error log file (`.err`). The old files are always replaced during the new compilation.

In addition, the resulting assembler file contains details in comments such as original names of objects stored in the register or memory cell.

PBCC Target code

See implementation documentation in Czech (details of the structure, naming conventions, data stack simulation, etc.).

Debug Support

In analogy with the compilation system phases, the debugging is performed in various levels too:

1. The debugging of SDCC Framework (Front-end) and of PicoBlaze port in MS Visual C Express
2. The analysis of iCodes in JSON format
3. The debugging of Back-end in Pharo Smalltalk (see `--dontClose`/`--dontProcess`)
4. The instruction-based software simulation (pBlazeIDE) and cycle-based hardware debugging (ModelSim) of the target code (PicoBlaze-3 Assembler)

Retargetability

According to developed methodology, we introduced few user-defined retargetability options (`--target` and `--dialect` arguments). For more details, see Czech documentation of SDCC and PBCC implementation.

KCPSM3 versus pBlazeIDE Assembler

Both dialects can be generated by PBCC. For more details, see PicoBlaze-3 Manual [2, page 69] or pBlazeIDE web page (http://www.mediatronix.com/Directives.htm).

Function Libraries

We have chosen a minimalistic subset of standard C library to support some basic functions such as character class tests `<ctype.h>`, time and date function `<time.h>`, basic mathematical functions `<math.h>`, and some utility functions `<stdlib.h>`.

We also make an abstraction of few processor-specific features (low-level functions) such as communication with ports (INPUT, OUTPUT), handling the interrupts (enable/disable), etc.

Low-level functions

Processor Interrupt `<intr.h>`:

```
void pbcc_set_interrupt_handler(void *(void))
void pbcc_set_interrupt(bool enable)
```

```
inline void pbcc_enable_interrupt()
inline void pbcc_disable_interrupt()
inline bool pbcc_enabled_interrupt()
```

Input/Output port <port.h>:

```
unsigned char pbcc_input_char(unsigned char port)
unsigned short pbcc_input_short(unsigned char port)
void pbcc_output_char(unsigned char port, unsigned char cdata)
void pbcc_output_short(unsigned char port, unsigned short sdata)
...
```

Delay <delay.h>:

```
void delay(unsigned char c);
```

Mathematical Functions <math.h>

Arguments and returned values are decimal types.

```
unsigned int pow(unsigned int, unsigned int);
unsigned int sqrt(unsigned int, unsigned int);
```

Integer Type Limits <limits.h>

The numbers given in parentheses are values for the PicoBlaze C Compiler/for typical 32-bit Unix system.

CHAR_BIT	bits in char	(8/8)
CHAR_MAX	max value of char	(+127/+127 or +255)
CHAR_MIN	min value of char	(–128/–128 or 0)
INT_MAX	max value of int	(+32 767/+32 767)
INT_MIN	min value of int	(–32 768/–32 768)
LONG_MAX	max value of long	(+2 147 483 647)
LONG_MIN	min value of long	(–2 147 483 648)
SCHAR_MAX	max value of signed char	(+127/+127)
SCHAR_MIN	min value of signed char	(–128/–128)
SHRT_MAX	max value of short	(+32 767/+32 767)
SHRT_MIN	min value of short	(–32 768/–32 768)
UCHAR_MAX	max value of unsigned char	(+255/+255)
UINT_MAX	min value of unsigned int	(+65 535/+65 535)
ULONG_MAX	max value of unsigned long	(+4 294 967 295/+4 294 967 295)
USHRT_MAX	min value of unsigned short	(+65 535/+65 535)

LCD Driver <lcd.h>

```
void LCD_init()
void LCD_clear()
void LCD_set_cursor()
macro: LCD_wr(arg)
```

As a part of this driver, we introduce subset of functions from Input/Output <stdio.h>.

© 2008-2009, Zbyněk Křivka
http://www.vlam.cz

Arithmetic Operations

Since Xilinx PicoBlaze-3 implements no multiplication or division operation, we deal with these operators in a special way. For more details, see report about the simulation of arithmetic operations in PicoBlaze and the providing support for external operations (in Czech).

Problems and errors

Please, feel free to contact us (main contact: krivka@fit.vutbr.cz) whenever you find an error or problem (please, first, consult SDCC and PBCC manuals).

Conclusion

As the project is run only by one person, it is already useful to do some basic programming in language C. Although we do not provide all features as old PCCOMP compiler yet, we are working on it. During 2009, we introduced several optimizations such as register allocation using coloring or analysis of inline assembler.

Acknowledgement

This work has been supported by the Czech Ministry of Education, Youth and Sports grant MŠMT 2C06008 "Virtual Laboratory of Microprocessor Technology Application" (visit website http://www.vlam.cz) and the Research Plan No. MSM 0021630528.

References

1. Xilinx. PicoBlaze Product Brief. [Online] [Visited: 2008/11/13] http://www.xilinx.com/bvdocs/ipcenter/data_sheet/picoblaze_productbrief.pdf.
2. Xilinx, PicoBlaze 8-bit Embedded Microcontroller: User Guide for Spartan-3, Spartan-6, Virtex-5, and Virtex-6 FPGAs, UG129 (v2.0, January 28, 2010) [Online] [Visited: 2010/01/30] http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf.
3. Mediatronix. pBlazeIDE version 3.74 Beta. [Online] [Visited: 2008/11/13] http://www.mediatronix.com/pBlazeIDE.htm.
4. Andrew W. Appel and Jens Palsberg: *Modern Compiler Implementation in Java*, Second Edition, Cambridge University Press, 2002, 501 p., ISBN: 0-5218-2060-X.
5. JSON. [Online] [Visited: 2008/12/22] http://www.json.org.
6. Pavel Křivánek, Zbyněk Křivka: Smalltalk Squeak. [Online] [Visited: 2008/12/22] http://www.squeak.cz.
7. SDCC Compiler User Guide, Revision 5340, 2008 [sdccman.pdf].
8. Ota Jirák, Zbyněk Křivka: Design and Implementation of Back-end for PicoBlaze C Compiler, In: *Proceedings of the IADIS International Conference Applied Computing* 2009, Rome, IT, IADIS Press, 2009, pp. 135-138, ISBN 978-972-8924-97-3.
9. Ota Jirák, Zbyněk Křivka: Simulation-based Debugging of 8-bit Softcore Processor, In: *Proceedings of* XXXI*th International Autumn Colloquium Advanced Simulation of Systems*, Ostrava, CZ, MARQ, 2009, pp. 68-73, ISBN 978-80-86840-47-5.
10. Pharo Open Source Smalltalk. [Online] [Visited: 2010/02/06] http://www.pharo-project.org.
11. Pavel Křivánek: PharoKernel – pharo. [Online] [Visited: 2010/02/06] http://code.google.com/p/pharo/wiki/PharoKernel.

© 2008-2009, Zbyněk Křivka
http://www.vlam.cz