PBCC:
PicoBlaze C Compiler
User Manual

Zbyněk Křivka
krivka@fit.vutbr.cz

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

Content

Revisions

No.	Date	Author	Revision description
1	2009-12-30	Z. Křivka	First draft
2	2010-03-31	Z. Křivka	Second draft (new SDCC front-end version 3)

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 and use of its C compiler (version 2). 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 framework SDCC 3.0 and higher focused on 8-bit CPUs. As this is a user manual, it describes the command line interface of new version of the compiler (abbrev. PBCCv2 or PBCC). For more details about the architecture and implementation of PBCC, see PicoBlaze C Compiler report in Czech.

Keywords

Compiler, parser, language C (ANSI-C), 8-bit processor, PicoBlaze, PBCCv2, KCPSM3, SDCC 3.0, front-end, back-end

Terminology

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

IS = Instruction-set

PBCC[v2] = PicoBlaze C Compiler [version 2]

SDCC = Small Device C Compiler

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 (so-called port) in SDCC and the documentation of this part is totally insufficient, we executed several experiments to explore SDCC's IR and documented the way of creation of new port in SDCC 3.0.

In the end, PicoBlaze C Compiler is composed of several chained command-line tools as shown in Figure 1.

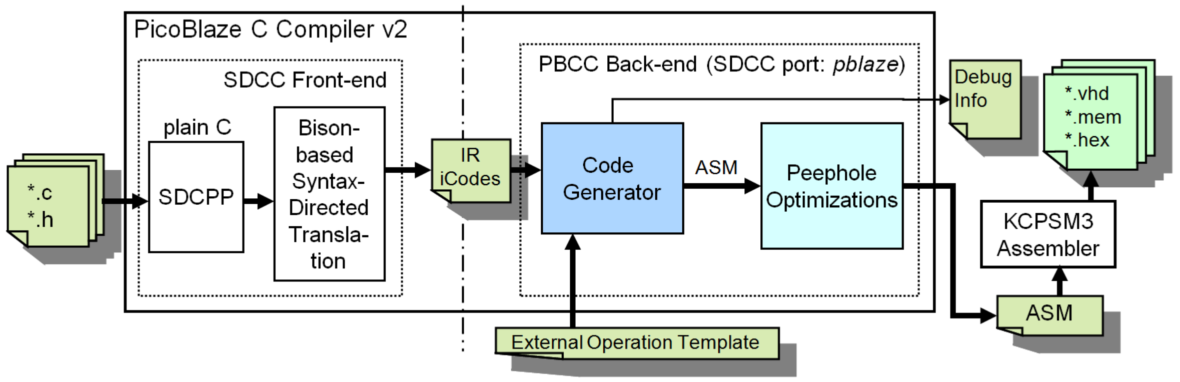

Figure 1. PicoBlaze C Compiler (PBCC) Architecture composing of front-end and back-end connected by IR in iCode format. For the classical list of compilation phases, see [4].

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.
- Due to missing native data stack, the scratchpad memory is used for the storage 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 also PBCCv2 supports source-code level inclusion only (no modularity).
- As there is no application binary interface by the manufacturer and no linkable code is generated, a 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 code 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 according to its conventions (see C-standard compliance in SDCC manual; see [**Chyba! Nenalezen zdroj odkazů.**]) 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 ([**Chyba! Nenalezen zdroj odkazů.**]) and website.

PBCC Back-end

The back-end internal architecture is based on the transformation of IR towards the optimized target code representation based on resource profiling information to handle essential requirements on the memory unassumingness. The SDCC's port is named *pblaze*.

Further, non-native operations (e.g. multiplication) are either (1) outsourced by external peripherals, or (2) simulated by PicoBlaze instruction set (memory consuming).

The resulting assembly code (ASM) is together with VHDL/Verilog templates compiled by KCPSM3 Assembler into the resulting hardware description of PicoBlaze processor 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 (for instance, using Xilinx ISE Webpack).

PBCC Usage

Primarily, PBCCv2 is designed as a command-line tool that can participate as a component of entire compilation system. The included batch files support Windows platform at the moment. We plan also to support the Unix-based platforms as the background tools and codes are multiplatform supporting both Microsoft Windows and Linux.

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 by SDCC containing PBCCv2 port called pblaze produces PicoBlaze-3 assembly code in the resulting .psm file.
2) The compilation from PicoBlaze-3 assembly language 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 1). In addition, we repeat the most useful arguments used by SDCC. Third, we introduce all supported and planned parameters of PBCCv2 back-end. 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 Webpack or QDevKit.

As the PicoBlaze processor is developed by Xilinx Company, we are focused on the FPGA chips also from Xilinx.

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"              ; directory of sdcc.exe and sdcpp.exe
@SET ASM_PATH="..\KCPSM3\Assembler"       ; directory of KCPSM3.exe
@SET SIMUL_PATH="..\pBlazeIDE"            ; 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 (`%SDCC_PATH%\source.psm` and `%ASM_PATH%\source.psm` if used `kcpsm3` argument).

PBCC Command-line Parameters

Now, we list the complete list of supported parameters of PBCC back-end to generate target code in PBCC compilation system. The arguments are case-sensitive. The syntax of the arguments has to be satisfied carefully; otherwise unexpected behavior (or error) can appear.

`-o <output.psm>`
Supported by SDCC: sets the output filename with the target code (PicoBlaze-3 Assembler) (Default: source.psm)

`-mpblaze`
chooses the target platform. Now, only `PicoBlaze3` processor is supported by the PBCCv2 port. For other target platforms, see SDCC manual. (Mandatory option)

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

`--verbose`
Supported by SDCC: generates even more detailed reports into log files including detailed comments directly into the resulting assembler with the emphasis on the human readability. (Default: omitted)

Future plans (Command-line Parameters)

NOT SUPPORTED: `--inline-asm-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: `--parse-inline-asm`
commands to the back-end to parse and check the validity of inlined assembler in C source code (Default: omitted)

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:).

1) PBCCv2/SDCC compiler execution (for more inspiration, see `pbcc.bat`):

```
C:\pbcc\sdcc\bin\sdcc.exe -c –IC:\pbcc\sdcc\device\include\picoBlaze
-mpicoBlaze --all-callee-saves <program.c>
```

2) 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:** As the import includes some text/instructions processing, 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 (3.0.1), we upgraded the whole front-end project to this version. Then, we introduce step-by-step tutorial how to compile the SDCC source codes including PicoBlaze-3 (pblaze) port in Cygwin at Microsoft Windows platform.

1) Run Cygwin bash shell and open directory with source codes of SDCC.
2) Generate configuration file (`configure` from `configure.in`):
   ```
   $ autoconf
   ```
3) Create all makefiles:
   ```
   $ ./configure --disable-pic16-port --disable-pic14-port --disable-json-port --disable-ds390-port --disable-ds400-port
   ```
4) Edit `src/SDCCmain.c` and comment out line `&json_port,`
5) Compile the code by execution of
   ```
   $ make all
   ```
6) Now, you test a command: `$ bin/sdcc.exe --version`
7) To use SDCC compiler, see SDCC manual [5].

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).

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 [**Chyba! Nenalezen zdroj odkazů.**]).

`__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)`
This keyword denotes the function that works as an interrupt handler. PicoBlaze processor supports only one interrupt, that is, `(?1)` parameter says if RETURNI is DISABLE (default behavior) or ENABLE (use non-zero constant).

Example:
```
void interrupt_handler(void) __interrupt (1)
{ /* the handler definition */ }
```

`__asm … __endasm;`
The pair of keywords encloses the inline assembler code (processor-specific). SDCC makes no validation of this code. PBCC does not check the syntax yet (planned feature) 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.

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 three levels:

1. The debugging of SDCC Framework (Front-end) and of PicoBlaze port in Eclipse CDT
2. The instruction-based software simulation (pBlazeIDE) of the target code (PicoBlaze-3 Assembler)
3. and cycle-based hardware debugging (ModelSim)

Retargetability

According to developed methodology, we introduced few user-defined retargetability options (`-mTARGET_PORT_NAME` from SDCC 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)
void pbcc_enable_interrupt()
void pbcc_disable_interrupt()
//BOOL pbcc_enabled_interrupt()
```

Input/Output port `<port.h>`:

```
unsigned char __port_read(unsigned char port)
//unsigned short pbcc_input_short(unsigned char port)
void __port_write(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)
void LCD_write(unsigned char)
```

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

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

Although the project is run only by one person, it is already useful to do some basic programming in language C. 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. During 2010, we upgraded the version of front-end to SDCC 3.0 and the back-end was completely rewritten in C language as the full-featured SDCC port of PicoBlaze 8-bit processor.

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).

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. SDCC Compiler User Guide, SDCC 3.0.1, Revision 6041, 2010. [Online] [Visited: 2010/03/31] http://sdcc.sourceforge.net/doc/sdccman.pdf.

6. 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.
7. 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.