Using PicoBlaze

buaa.byl@gmail.com

January 26, 2013

Contents

Chapter 1

Introduction

PicoBlaze®是*Xilinx*®针对低端应用开发的8位处理器，开放源代码。

1.1 KCPSM3

PicoBlaze另一个名称KCPSM，是Constant(K) Coded Programmable State Machine的简称，也即常量编码状态机。由Ken Chapman开发，所以也有人这么理解Ken Chapman's PSM。

本文分析的主要是Spartan3系列的版本，即KCPSM3，典型的KCPSM3占用96个Slices，等价于低端XC3S200器件5%的资源。

PicoBlaze是8位处理器，使用18位指令集，每个指令执行时间为2个周期。KCPSM3直接用xilinx原语编写，只有汇编器。这样就导致在使用中有些问题，其一汇编维护起来太麻烦，其次想添加新的指令很难。

而PicoBlaze的只有一份很旧的设计说明，针对VertixII器件的“Creating Embedded Microcontrollers (Programmable State Machines)”，缺少一个完整的设计文档，当然也没找到KCPSM3的设计说明。想要修改它非常不容易，所以我准备阅读它的源码，归纳设计的方法。虽然已经有PacoBlaze[1]，但想要方便以后的使用，还是自己阅读源码，并改写为Verilog。也是为了开阔自己的思路，提高能力。

[1] PacoBlaze：一个基于PicoBlaze的开源实现

1.2 Application of PicoBlaze

1. LED flasher.
2. PWM control and even generation.
3. Switch monitor.
4. UART interface and simple command/status terminal.
5. LCD character module display interface and control.
6. SPI master
7. I2C master.
8. Calculator.
9. Audio DSP processor.
10. DTMF tone telephone dialer including sine wave generation.
11. System monitoring.
12. Motor control.
13. Rotary encoder interface.
14. Calculator for frequency synthesizer.
15. Calculator for filter coefficient generation.
16. Emulation of a different micro controller.
17. PID control.
18. Mouse/Keyboard interface.
19. Keypad scanner.
20. Power supply monitoring and control.
21. Servo control.
22. Built-in test equipment.
23. Configuration management.
24. Design Authentication Processor.
25. Implementing peripherals for MicroBlaze or PPC.
26. Interrupt controller for MicroBlaze or PPC.

1.3 Code style

c code style:

```c
int adder(int a, int b)
{
    return a + b;
}
```

verilog code style:

```
module adder(input a,
        input b,
        output c);

assign c = a + b;

endmodule
```

Part I

Using KCPSM3

Chapter 2

How to store code

Chapter 3

What assembler generated

Chapter 4

Instruction set

Chapter 5

Assembler in python

Part II

Reading KCPSM3

Chapter 6

Architecture

Chapter 7

Interrupt

阅读代码需要一个开始的地方，因为PicoBlaze定位是可编程状态机，那么就从外部中断开始阅读，中断包含同步、使能、上下文切换。

先介绍同步模块“interrupt_capture”。

7.1 int_capture_flop

我们先从外部中断引脚开始看，看看经过哪些器件。以下是picoblaze的顶层接口：

```
1  module kcpsm3(
2          address,
3          instruction,
4          port_id,
5          write_strobe,
6          out_port,
7          read_strobe,
8          in_port,
9          interrupt,
10         interrupt_ack,
11         reset,
12         clk) ;
13
14 output  [9:0]   address ;
15 input   [17:0]  instruction ;
16 output  [7:0]   port_id ;
17 output          write_strobe, read_strobe, interrupt_ack ;
18 output  [7:0]   out_port ;
19 input   [7:0]   in_port ;
20 input           interrupt, reset, clk ;
```

可以从字义上理解interrupt为外部中断的入口，而interrupt_ack为中断确认信号，interrupt首先连入int_capture_flop逻辑：

```
// Interrupt capture
FDR int_capture_flop (
    .D(interrupt),
    .Q(clean_int),
    .R(internal_reset),
    .C(clk));
```

这个FDR是“同步复位D触发器”原语。FDR是比较好理解的，D是输入，Q是输出，R是复位，C是时钟。
在C:\Xilinx\12.4\ISE_DS\ISE\verilog\xeclib\unisims下是官方给出的仿真代码。在后面还会涉及到LUT（查找表），所以在这里先介绍一下。

FDR

```
module FDR (Q, C, D, R);
    parameter INIT = 1'b0;
    output Q;
    reg    Q;
    input  C, D, R;
    always @(posedge C)
        if (R)
        Q <= 0;
        else
        Q <= D;
endmodule
```

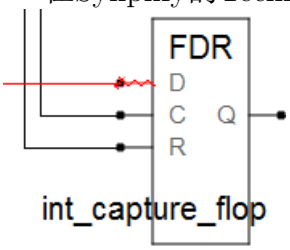
LUT4

```
module LUT4 (O, I0, I1, I2, I3);
    parameter INIT = 16'h0000;
    input I0, I1, I2, I3;
    output O;
    wire out0, out1, out2, out3, out;
    assign O = INIT[{I3,I2,I1,I0}];
endmodule
```

可以看出就是一个16bit的ROM结构，O是输出，I3、I2、I1、I0是地址线。因为原语不太直观，所以在这里把int_capture_flop改写为rtl描述。

```
always@(posedge clk)
begin
    if (internal_reset)
        clean_int <= 1'b0;
    else
        clean_int <= interrupt;
end
```

在Synplify的Technology View可以看到如下的图



这个逻辑同步外部的中断信号。interrupt是外部中断信号，`clean_int`是同步后的中断信号。`internal_reset`是PicoBlaze内部复位信号，当处理器复位的时候，`internal_reset`会持续一段时间。

用原语来写有个好处就是综合的结果基本上就是你想要的电路，但是代价就是不容易维护，也不利于别人阅读。

7.2 Karnaugh Graphic

用原语来写就涉及到自己优化逻辑的问题，就需要理解卡诺图了，而阅读用原语写的组合逻辑代码就更痛苦了，因为LUT的值是一个16位的数字，很不直观这就要自己写个脚本把16位的值转为一个平面的图。

karnaugh.py

```
#!/bin/python
import sys

def itob(a, bitnum=16):
    bits = []
    for i in range(0, bitnum):
        if (a & (1 << (bitnum - 1 - i))) != 0:
            bits.append('1')
        else:
            bits.append('0')
    return bits

def printv(bits):
    #dec format index
    sys.stdout.write('bitno: ')
    for i in range(15, -1, -1):
        sys.stdout.write('%4d' % i)
        if (i % 4) == 0:
            sys.stdout.write(' | ')
        else:
            sys.stdout.write(' ')
    sys.stdout.write('\n')

    #binary format index
    sys.stdout.write('bitno: ')
    for i in range(15, -1, -1):
        sys.stdout.write(''.join(itob(i, bitnum=4)))
        if (i % 4) == 0:
            sys.stdout.write(' | ')
        else:
            sys.stdout.write(' ')
    sys.stdout.write('\n')

    #value
    sys.stdout.write('value: ')
    bitno = 15
    for bit in bits:
        sys.stdout.write('%4s' % bit)
        if (bitno % 4) == 0:
            sys.stdout.write(' | ')
        else:
```

```
42              sys.stdout.write(' ')
43          bitno -= 1
44      sys.stdout.write('\n')
45      sys.stdout.write('\n')
46
47      #karnaugh map
48      print('kano-graphic:')
49      sys.stdout.write('||= bit(3210) =|' + \
50              '|=  xx11 =||=  xx10 =||=  xx00 =||=  xx01 =||\n')
51
52      sys.stdout.write('||         11xx||%7s  ||%7s  ||%7s  ||%7s  ||\n' % (
53          bits[0*4 + 0], bits[0*4 + 1], bits[0*4 + 3], bits[0*4 + 2]))
54
55      sys.stdout.write('||         10xx||%7s  ||%7s  ||%7s  ||%7s  ||\n' % (
56          bits[1*4 + 0], bits[1*4 + 1], bits[1*4 + 3], bits[1*4 + 2]))
57
58      sys.stdout.write('||         00xx||%7s  ||%7s  ||%7s  ||%7s  ||\n' % (
59          bits[3*4 + 0], bits[3*4 + 1], bits[3*4 + 3], bits[3*4 + 2]))
60
61      sys.stdout.write('||         01xx||%7s  ||%7s  ||%7s  ||%7s  ||\n' % (
62          bits[2*4 + 0], bits[2*4 + 1], bits[2*4 + 3], bits[2*4 + 2]))
63      sys.stdout.write('\n')
64
65  def printlut4(a):
66      bits = itob(a)
67      print('original: 0x%04X' % a)
68      print
69      printv(bits)
70
71  if __name__ == '__main__':
72      if len(sys.argv) == 1:
73          print 'need hex digit!'
74      else:
75          printlut4(int(sys.argv[1], 16))
```

对于0x1010，转换的结果如下：

```
1  kano-graphic:
2  ||= bit(3210) =||=  xx11 =||=  xx10 =||=  xx00 =||=  xx01 =||
3  ||         11xx||      0  ||      0  ||      1  ||      0  ||
4  ||         10xx||      0  ||      0  ||      0  ||      0  ||
5  ||         00xx||      0  ||      0  ||      0  ||      0  ||
6  ||         01xx||      0  ||      0  ||      1  ||      0  ||
```

Chapter 8

Interrupt Logic

Chapter 9

Flow Control

Chapter 10

PC and Registers

Chapter 11

Store Memory

Chapter 12

Call Stack

Chapter 13

ALU and Multiplexer

Chapter 14

Input and output

Chapter 15

Rewrite KCPSM3 in verilog

Chapter 16

Assembler preprocessor in python

Part III

JTAG

Chapter 17

Missing debug

Chapter 18

What is jtag

Chapter 19

Jtag slave in verilog

Chapter 20

Jtag master in c

Chapter 21

A stm32-based jtag master

Chapter 22

A kcpsm3-base jtag master

Chapter 23

Add breakpoint controler

Chapter 24

Add boundary scan to kcpsm3

Chapter 25

Reuse lattice mico8 gcc

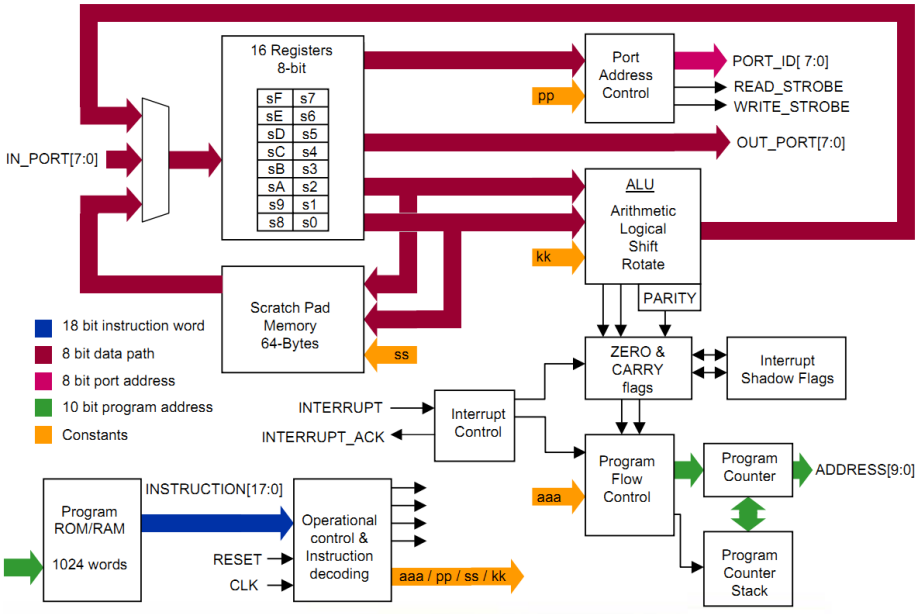
Part IV

Port GCC

Chapter 26

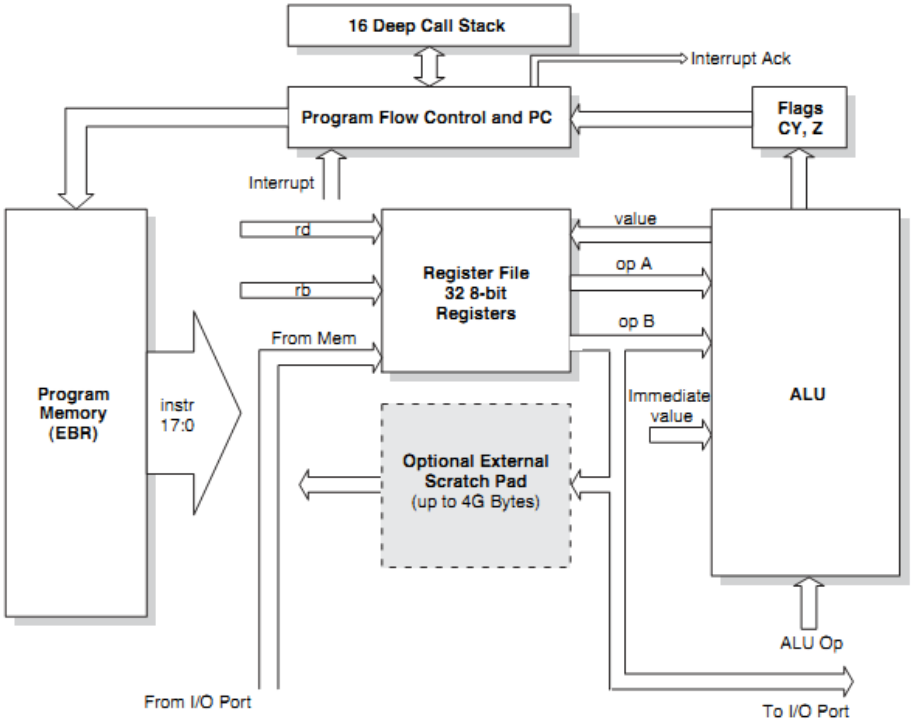
Learning

Xilinx有PicoBlaze，而Lattice有一个8位的处理器Mico8。这个是PicoBlaze的系统框架图[1]



[1]图片来自PicoBlaze Manual.pdf，Page8

这个是Mico8的系统框架图[2]



从系统框图上看，Mico8和PicoBlaze的设计很相似。更让人惊喜的是Lattice移植了Mico8版本的GCC，这样意味着很有可能把这个GCC修改一下支持PicoBlaze！并且也可以学习移植GCC，毕竟x86、arm和sparc都不简单。

[2]图片来自rd1026.pdf，Page1

26.1 Compare

既然要想弄明白Lattice是怎么移植GCC的，那么最好的办法就是找出lm8-gcc和原始的gcc之间的差异，然后对比差异来学习。

这么一来第一件事就是找到两份源代码，一个是lm8-gcc，另一个是原始的gcc。在Lattice找到了LatticeMico8_Tools_v3_15的链接，下载回来是gcc-lm8-2010-09-28.tar.bz2，这个就是移植Mico8的gcc。解压后可以知道这个gcc-lm8是以GCC 4.4.3为基础改出来的。那么我们再到GCC的ftp把gcc-4.4.3.tar.bz2 下载回来。

我把lm8-gcc的源码解压为gcc-4.4.3-lm8，把原始的gcc解压为gcc-4.4.3-org。需要对比差异，最简单的就是使用diff工具，在linux下这个工具常用来生成patch文件。

至于diff怎么用，直接输入以下命令：

```
#/bin/sh
diff --help
```

输出如下：

```
Usage: diff [OPTION]... FILE1 FILE2

  -i  --ignore-case  Consider upper- and lower-case to be the same.
  -w  --ignore-all-space  Ignore all white space.
  -b  --ignore-space-change  Ignore changes in the amount of white space.
  -B  --ignore-blank-lines  Ignore changes whose lines are all blank.
  -I RE  --ignore-matching-lines=RE  Ignore changes whose lines all match RE.
  --binary  Read and write data in binary mode.
  -a  --text  Treat all files as text.

  -c  -C NUM  --context[=NUM]  Output NUM (default 2) lines of copied context.
  -u  -U NUM  --unified[=NUM]  Output NUM (default 2) lines of unified context.
    -NUM  Use NUM context lines.
    -L LABEL  --label LABEL  Use LABEL instead of file name.
    -p  --show-c-function  Show which C function each change is in.
    -F RE  --show-function-line=RE  Show the most recent line matching RE.
  -q  --brief  Output only whether files differ.
  -e  --ed  Output an ed script.
  -n  --rcs  Output an RCS format diff.
  -y  --side-by-side  Output in two columns.
    -w NUM  --width=NUM  Output at most NUM (default 130) characters per line.
    --left-column  Output only the left column of common lines.
    --suppress-common-lines  Do not output common lines.
  -DNAME  --ifdef=NAME  Output merged file to show ‘#ifdef NAME’ diffs.
  --GTYPE-group-format=GFMT  Similar, but format GTYPE input groups with GFMT.
  --line-format=LFMT  Similar, but format all input lines with LFMT.
  --LTYPE-line-format=LFMT  Similar, but format LTYPE input lines with LFMT.
    LTYPE is ‘old’, ‘new’, or ‘unchanged’.  GTYPE is LTYPE or ‘changed’.
    GFMT may contain:
      %<  lines from FILE1
```

```
      %>  lines from FILE2
      %=  lines common to FILE1 and FILE2
      %[-][WIDTH][.[PREC]]{doxX}LETTER  printf-style spec for LETTER
        LETTERs are as follows for new group, lower case for old group:
          F  first line number
          L  last line number
          N  number of lines = L-F+1
          E  F-1
          M  L+1
    LFMT may contain:
      %L  contents of line
      %l  contents of line, excluding any trailing newline
      %[-][WIDTH][.[PREC]]{doxX}n  printf-style spec for input line number
    Either GFMT or LFMT may contain:
      %%  %
      %c'C'  the single character C
      %c'\OOO'  the character with octal code OOO

  -l  --paginate  Pass the output through `pr' to paginate it.
  -t  --expand-tabs  Expand tabs to spaces in output.
  -T  --initial-tab  Make tabs line up by prepending a tab.

  -r  --recursive  Recursively compare any subdirectories found.
  -N  --new-file  Treat absent files as empty.
  -P  --unidirectional-new-file  Treat absent first files as empty.
  -s  --report-identical-files  Report when two files are the same.
  -x PAT  --exclude=PAT  Exclude files that match PAT.
  -X FILE  --exclude-from=FILE  Exclude files that match any pattern in FILE.
  -S FILE  --starting-file=FILE  Start with FILE when comparing directories.

  --horizon-lines=NUM  Keep NUM lines of the common prefix and suffix.
  -d  --minimal  Try hard to find a smaller set of changes.
  -H  --speed-large-files  Assume large files and many scattered small changes.

  -v  --version  Output version info.
  --help  Output this help.

If FILE1 or FILE2 is `-', read standard input.
```

这个说明是很详细的，基本上不用再多说什么了。

我使用以下的命令对比。参数u是添加行号；N是如果文件是新增的，那么把源文件当成空的；r是递归调用。（由于GCC源码太大，对比的时间很长，耐心等候吧）

```
#/bin/sh
diff -uNr gcc-4.4.3-org gcc-4.4.3-lm8 > result.diff
```

得到差异结果有130Kb，不方便阅读，也不方便贴出来，所以自己写个简单的python脚本提取文件名。这样就相当于有了一个索引，阅读和定位都方便很多。

diffbrief.py

```
#!/bin/python
import sys, re

fin = open(sys.argv[1])
lines = fin.read()
fin.close()

regex_prefix = re.compile(r'^diff -uNr ')
regex_header = re.compile(r'^diff -uNr ([^ ]+) ([^ ]+)')
lineno = 1

for line in lines.split('\n'):
    if regex_prefix.match(line):
        res = regex_header.search(line)
        relative_path = res.groups()[0]
        relative_path = '/'.join(relative_path.split('\\')[1:])
        print '%04d: %s' % (lineno, relative_path)
    lineno += 1
```

输出的结果如下，其中左边的数字是result.diff的行号，右边的是文件位置：

```
0001: config.sub
0032: configure
0045: configure.ac
0058: gcc/config/lm8/constraints.md
0087: gcc/config/lm8/crt0.S
0177: gcc/config/lm8/libgcc.S
1878: gcc/config/lm8/lm8-protos.h
1951: gcc/config/lm8/lm8.c
3795: gcc/config/lm8/lm8.h
4427: gcc/config/lm8/lm8.md
5082: gcc/config/lm8/lm8.opt
5127: gcc/config/lm8/predicates.md
5192: gcc/config/lm8/t-lm8
5261: gcc/config/gcc
5284: libgcc/config/lm8/t-default
5291: libgcc/config.host
```

26.2 Summary

对照源码的目录之后知道，gcc/config/lm8 和libgcc/config/lm8 这两个目录是不存在的，所以可以总结出对比的结果：

1. 新文件
    - gcc/config/lm8/constraints.md
    - gcc/config/lm8/crt0.S
    - gcc/config/lm8/libgcc.S
    - gcc/config/lm8/lm8-protos.h
    - gcc/config/lm8/lm8.c
    - gcc/config/lm8/lm8.h
    - gcc/config/lm8/lm8.md
    - gcc/config/lm8/lm8.opt
    - gcc/config/lm8/predicates.md
    - gcc/config/lm8/t-lm8
    - libgcc/config/lm8/t-default
2. 修改
    - config.sub
    - configure
    - configure.ac
    - config.gcc
    - libgcc/config.host

26.3 config.sub

```
+        lm8 | lm8-*)
+                basic_machine=lm8-unknown
+                ;;
         os400)
                basic_machine=powerpc-ibm
                os=-os400
@@ -1508,6 +1512,9 @@
                ;;
         or32-*)
                os=-coff
+                ;;
+        lm8-*)
+                os=-elf
                ;;
         *-tti)        # must be before sparc entry or we get the wrong os.
                os=-sysv3
```

Chapter 27

How lm8-gcc porting

Chapter 28

Port kcpsm3

Part V

SPI and UART

Chapter 29

Using kcuart

Chapter 30

Implement a SPI Controler

Part VI

Web Server

Part VII

RTOS

Part VIII

SDRAM