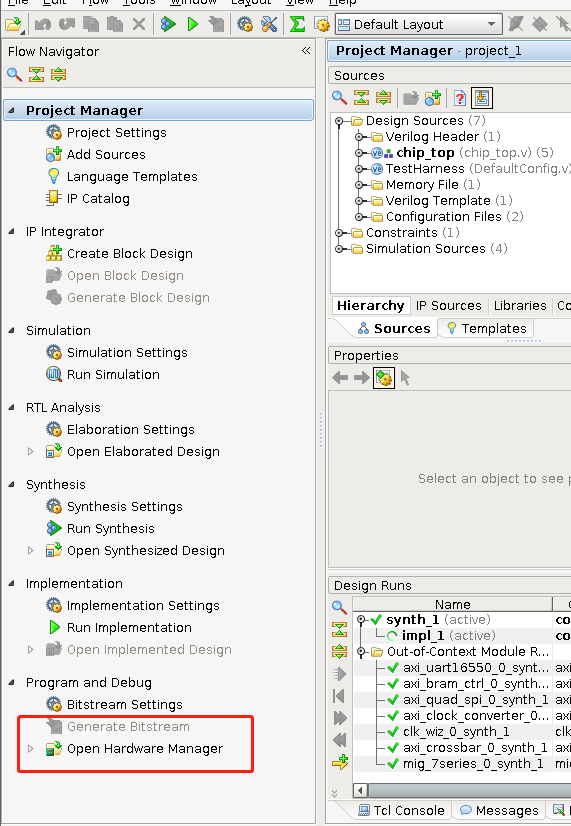
项目参考

<https://github.com/eugene-tarassov/vivado-risc-v>

<https://github.com/lowRISC/lowrisc-chip/tree/debug-v0.3>

<https://github.com/Compass-All/Raft>

### Vivado



一般只需要generate bitstream和open hardware manager就可以使用了

open hardware manager就是打开可以烧录的界面 点tools-auto connect

#### 烧录flash

烧写mcf

1.首先bit转mcs

write\_cfgmem -format mcs -size 128 -interface BPIx16 -loadbit {up 0x00000000 "/home/yy/Desktop/fpga-rocket-chip/project\_1/project\_1.runs/impl\_1/chip\_top.bit" } -file "/home/yy/Desktop/fpga-rocket-chip/project\_1/project\_1.runs/impl\_1/suibian3.mcs"

/home/yy/Desktop/fpga-rocket-chip/project\_1/project\_1.runs/impl\_1/chip\_top.bit

绝对地址就行了

"/home/yy/Desktop/fpga-rocket-chip/project\_1/project\_1.runs/impl\_1/suibian3.mcs"

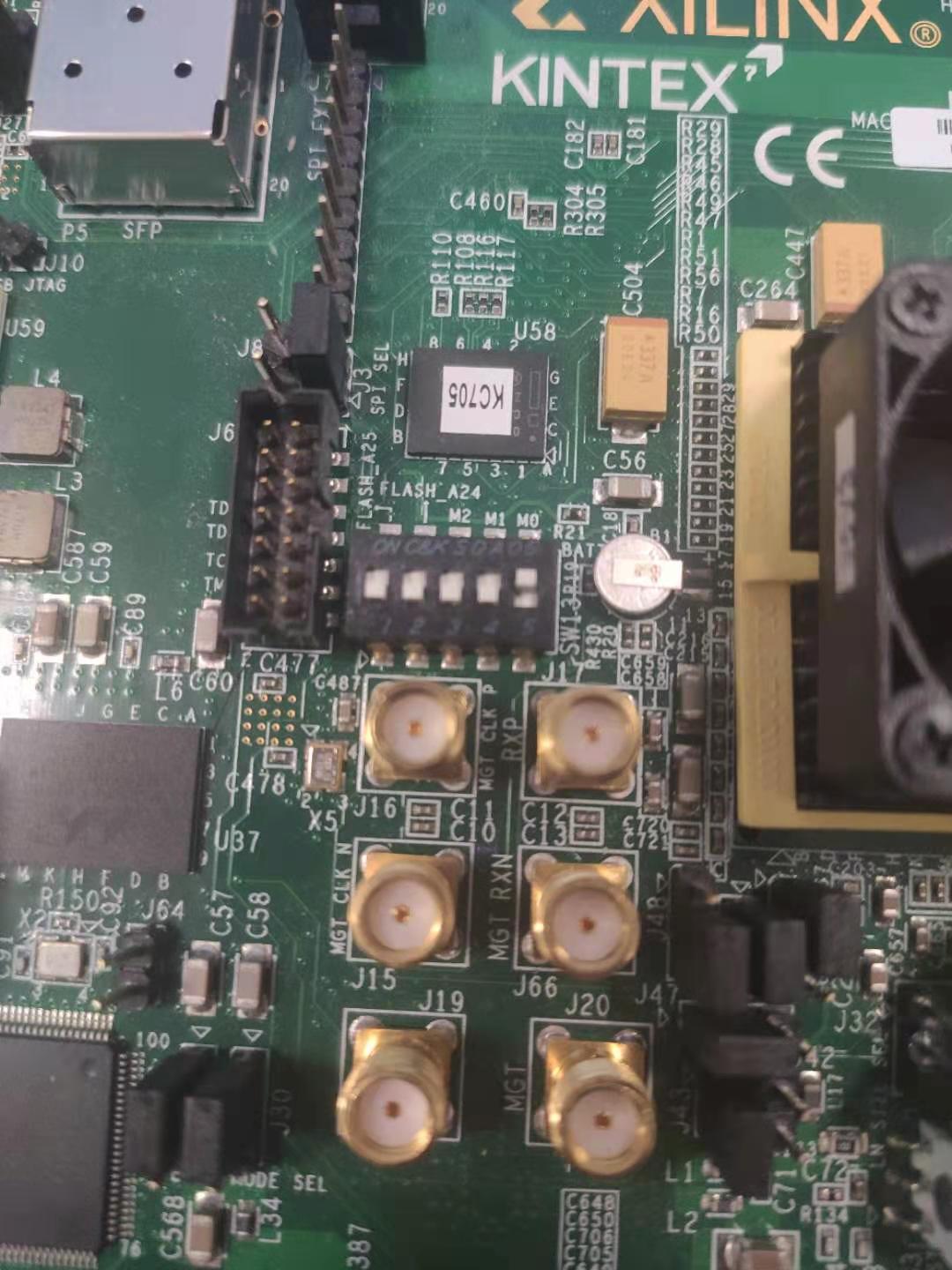
suibian3.mcs烧写出来的名字随便命名

这些参数都是查ug810\_KC705\_Eval\_Bd出来的 -size 128 -interface BPIx16

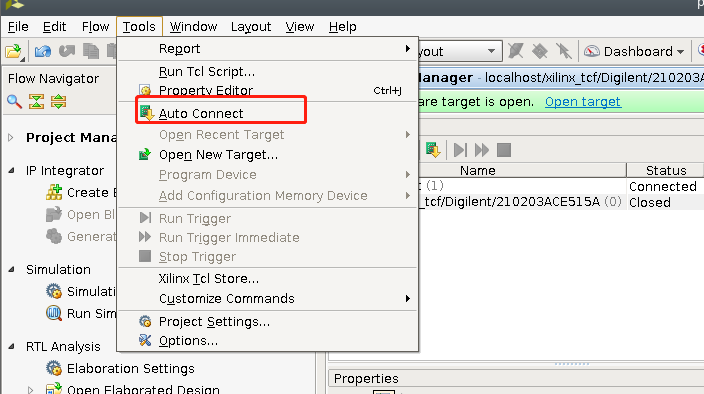
The Linear BPI flash memory located at U58 provides 128 MB of nonvolatile storage that can be used for configuration or software storage. The data, address, and control signals are connected to the FPGA. The BPI flash memory device is packaged in a 64-pin BGA.

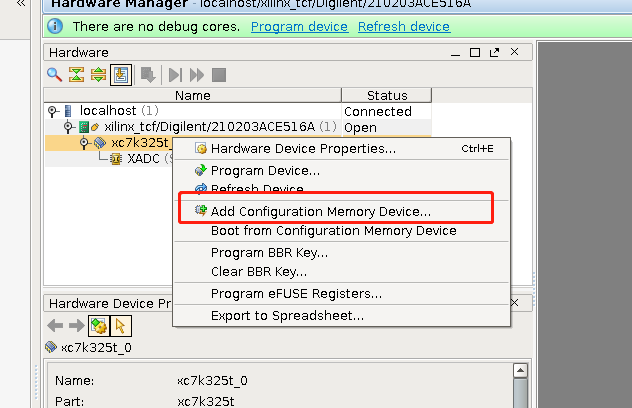
• Part number: PC28F00AP30TF (Micron) • Supply voltage: 2.5V • Datapath width: 16 bits (26 address lines and 7 control signals) • Data rate: Up to 33 MHz

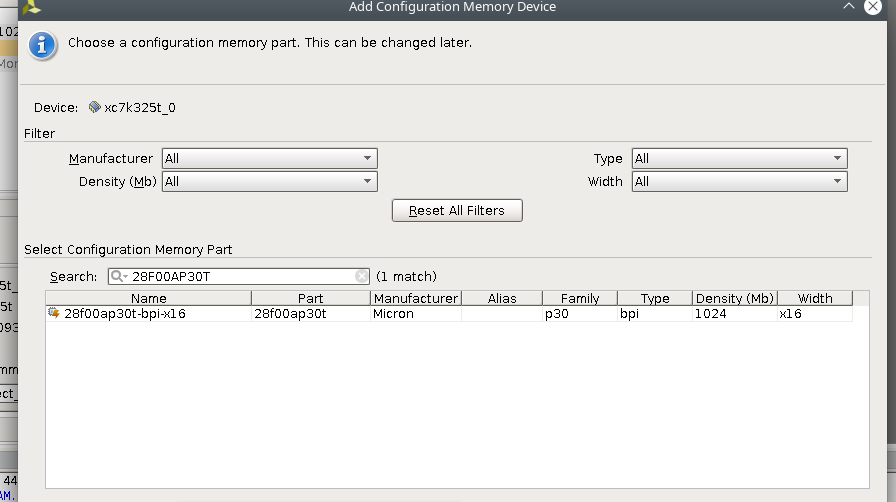
2.然后按钮4开（这个图是按钮5开）

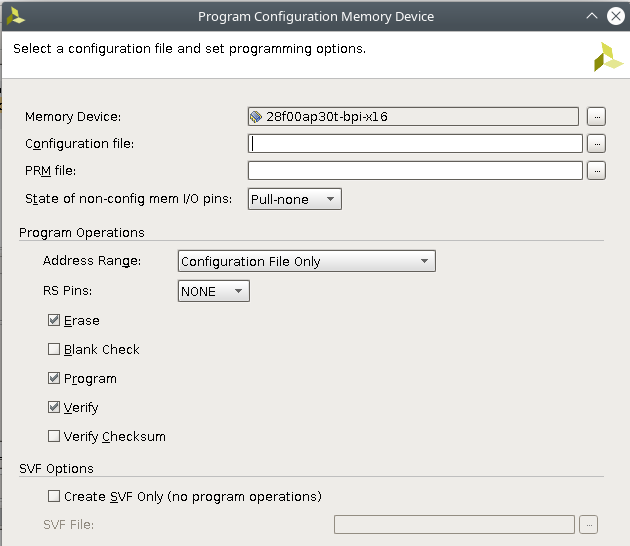


choose 28F00AP30T









#### Fpga-rocketchip ip核的配置

Ddr3 IP 核的配置 debug mode 一定要关掉

axi （这个不是第一页就是第二页）选中

时钟5000ps

还有个resetmode我记得应该是high 反正就两个选项你调一下

AXI CLK Frequency - 60 MHz

#### Vivado驱动安装

Uart

https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers

Jtag

[vivado\_ubuntu\_JTAG驱动安装](http://training.eeworld.com.cn/video/15232)

http://training.eeworld.com.cn/video/15232

### Rocketchip相关

#### Rocketchip可以配置多少g内存

class WithExtMemSize(n: BigInt) extends Config((site, here, up) => {

fix type mismatch when using WithExtMemSize #2671 搜一下这个

在subsystem的config里面

修改

-class WithExtMemSize(n: Long) extends Config((site, here, up) => {

+class WithExtMemSize(n: BigInt) extends Config((site, here, up) => {

在system的config里面

new WithExtMemSize(x"4000\_0000") ++ 就是说他能配1g内存

Kc705可以这么配

#### Rocc处理器linux kernel 和bootloader都要改让他支持自定义指令

最重要的是sstatus寄存器 不改他的话他会有illegal instruction

https://github.com/riscv/riscv-binutils-gdb/issues/156

具体原因

You would need to modify Linux to set the XS bits and save/restore any RoCC state you have in the context switch code for Linux. The source of the illegal instruction is likely due to the XS bits being zero, which is analogous to having the FS bits set to zero and trying to run F/D instructions.

Take a look at how the FPU state is saved/restored in arch/riscv/kernel/entry.S. An initial (not thread safe, not secure) hack would just be to set the XS bits blindly to 1.

I don't think that this is related to binutils, though. There shouldn't be any need to recompile the kernel with binutils understanding customX. I may be missing something, though.

mstatus（Machine Status）它保存全局中断使能，以及许多其他的状态，如图 10.4 所示。

void start\_thread(struct pt\_regs \*regs, unsigned long pc,

unsigned long sp)

{

regs->sstatus = SR\_SPIE| SR\_XS\_INITIAL;

if (has\_fpu)

regs->sstatus |= SR\_FS\_INITIAL;

regs->sepc = pc;

regs->sp = sp;

set\_fs(USER\_DS);

}

#### Rocketchip emulator （用来debug）

./emulator-freechips.rocketchip.system-DefaultConfig-debug +verbose /home/yy/riscv-tests/benchmarks/hello.riscv 2>&1 | tee run.log

会输出到屏幕有点烦

./emulator-freechips.rocketchip.system-DefaultConfig-debug +verbose /home/yy/riscv-tests/benchmarks/hello.riscv 2> run.log

不输出到屏幕

但是好处是可以在代码写printf

蓝色是binary

看波形方法

./emulator-freechips.rocketchip.system-DefaultConfig-debug +max-cycles=100000000 --vcd=helloworld.riscv.vcd /home/yy/riscv-tests/benchmarks/hello.riscv

要先make debug

Gdb方法

1. openocd安装

git clone <https://github.com/riscv/riscv-openocd.git>

cd riscv-openocd

./bootstrap (when building from the git repository)

./configure [options]

make

sudo make install

1. <https://github.com/chipsalliance/rocket-chip>教程基本在这

Config里面增加红色那玩意

class DefaultConfig extends Config( new WithJtagDTM ++ new WithNBigCores(1) ++ new WithCoherentBusTopology ++ new BaseConfig)

这样的话就make就行

或者在src/main/scala/system/Configs.scala增加:

class DefaultConfigRBB extends Config(

new WithJtagDTMSystem ++ new WithNBigCores(1) ++ new WithCoherentBusTopology ++ new BaseConfig)

class QuadCoreConfigRBB extends Config(

new WithJtagDTMSystem ++ new WithNBigCores(4) ++ new WithCoherentBusTopology ++ new BaseConfig)

CONFIG=freechips.rocketchip.system.DefaultConfigRBB make debug

1. We suppose that helloworld is our program, you can use crt.S, syscalls.c and the linker script test.ld to construct your own program, check examples stated in [riscv-tests](https://github.com/riscv/riscv-tests).就是那个在riscv-test下编译
2. 在一个shell开

./emulator-freechips.rocketchip.system-DefaultConfigRBB +jtag\_rbb\_enable=1 --rbb-port=9823 helloworld

会显示

This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag\_rbb\_enable=1.

Listening on port 9823

Attempting to accept client socket

5.新开一个shell 新加一个文件 cemulator.cfg

里面写

interface remote\_bitbang

remote\_bitbang\_host localhost

remote\_bitbang\_port 9823

set \_CHIPNAME riscv

jtag newtap $\_CHIPNAME cpu -irlen 5

set \_TARGETNAME $\_CHIPNAME.cpu

target create $\_TARGETNAME riscv -chain-position $\_TARGETNAME

gdb\_report\_data\_abort enable

init

halt

1. 在刚才那个shell

openocd -f ./cemulator.cfg

1. 再开一个shell

riscv64-unknown-elf-gdb helloworld

这个helloworld记得用 -g -Og 编译

比如

riscv64-unknown-elf-gcc -I./../env -I./common -I./hello -I./median -I./qsort -I./rsort -I./towers -I./vvadd -I./multiply -I./mm -I./dhrystone -I./spmv -I./mt-vvadd -I./mt-matmul -I./pmp -DPREALLOCATE=1 -mcmodel=medany -static -std=gnu99 -O2 -ffast-math -fno-common -fno-builtin-printf -fno-tree-loop-distribute-patterns -o hello.riscv ./hello/komodo.c ./hello/hello1.c ./common/syscalls.c ./common/crt.S -static -nostdlib -nostartfiles -lm -lgcc -T ./common/test.ld -g -Og

然后

(gdb) set remotetimeout 3000（这个挺重要的 2000感觉不够 我设的3000）

(gdb) target remote localhost:3333

Remote debugging using localhost:3333

0x0000000000010058 in ?? ()

(gdb) load

Loading section .text.init, size 0x294 lma 0x80000000

Loading section .tohost, size 0x48 lma 0x80001000

Loading section .text, size 0xeb4 lma 0x80002000

Loading section .rodata.str1.8, size 0xa8 lma 0x80002eb8

Loading section .rodata, size 0x158 lma 0x80002f60

Start address 0x80000000, load size 5008

Transfer rate: 83 bytes/sec, 1001 bytes/write.

(gdb) x 0x80000000

0x80000000 <\_start>: 0x00000093

(gdb)

### Verilator安装

<https://chipyard.readthedocs.io/en/latest/Chipyard-Basics/Initial-Repo-Setup.html>