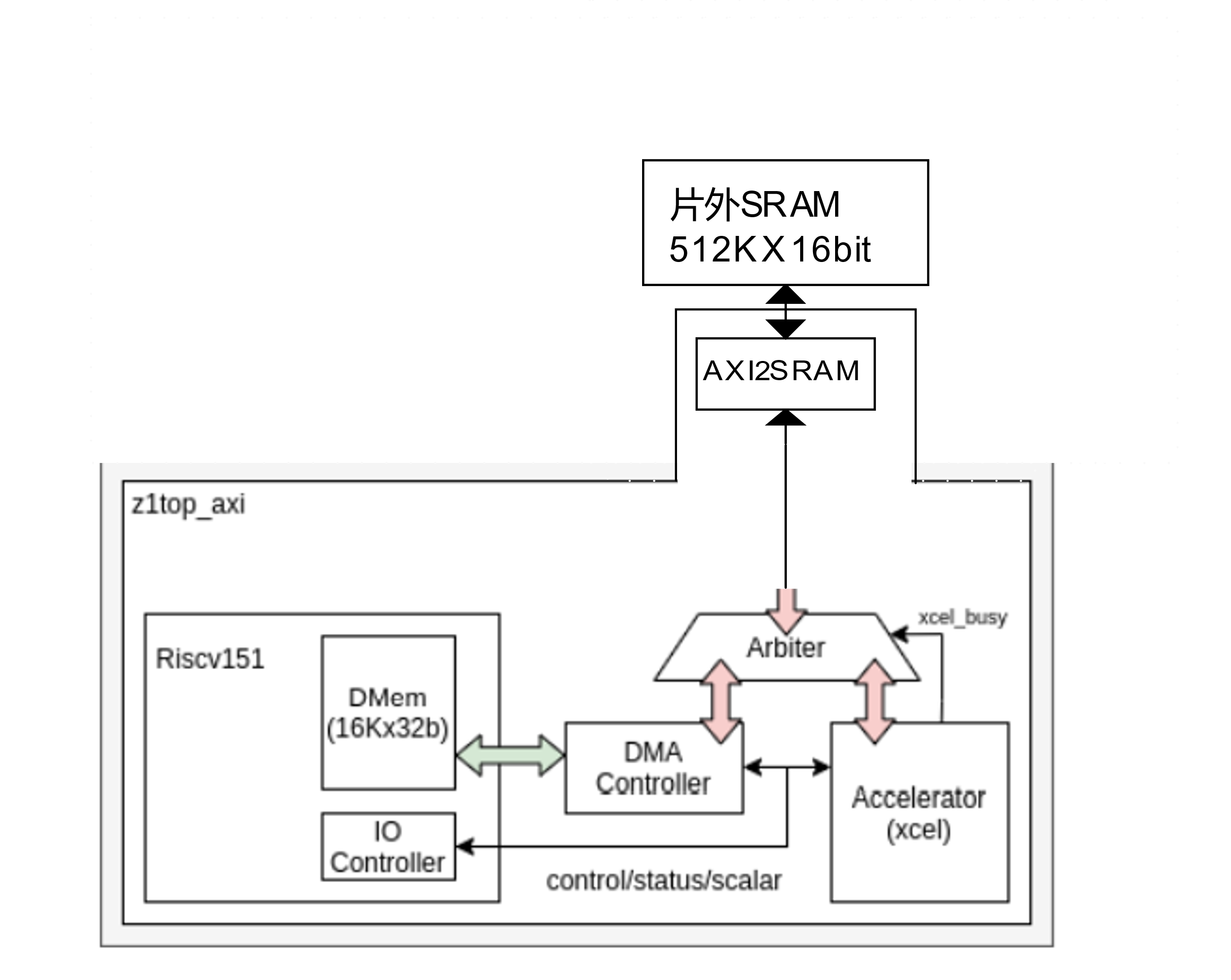
## 简介

根据EECS151 FPGA的课程设计要求, 设计和实现了一个三级六流水线的RISC-V CPU并将和UART集成。进一步集成CNN加速器和DMA，系统中使用AXI总线进行数据交互。因为没有实验要求的PYNQ板子，设计在EGO1 FPGA上实现, 同时课程设中的DRAM也改成了片外的SRAM。下面是整个系统框图的overview。



## Design Log

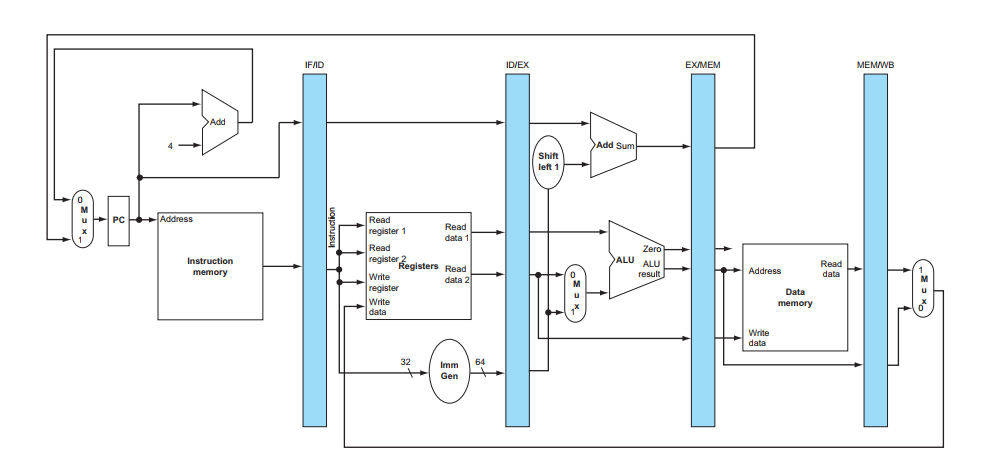
**CPU如何划分pipeline?**

三级流水线划分方式为: IF | ID&EX&MEM | WB，因为IMEM和DMEM都是同步读写的sram，他们相当于插入了一个寄存器，这样我们不需要再另外插入寄存器(其实还是要插一些信号的寄存器的)就可以实现三级流水线，详细的框图画在纸上了。

Stage 3

Stage 2

Stage 1



CPU已经完成，目前运行矩阵乘法程序benchmark的CPI约为1.18, 频率约为70Mhz.

**下一步?**

把原本提供的加速器和DMA集成到系统中，进行推理，把DRAM改为SRAM在EGO1上实现预测。先集成，修改CPU接口!

**CPU接口已经完成, 下一步写axi2sram(FPGA), 然后验证axi c程序。**

已经验证完成axi\_test.c程序。

**移植Lenet c代码, 跑最sw的cnn和naïve hw的cnn。**

CPU跑CNN预测目前没有太大问题，但利用xcel加速器运行CNN预测的时候结果不对，不知道是axi2sram接口的原因，还是什么原因。

课程project提供的xcel加速器已经运行正确，之前运行不正确的原因是之前软件分配给第一层卷积层输出数据的buffer太小。

**开发环境笔记**

## Dev environment

1. Diagram: pencil and paper

2. OS: Windows

3. gcc tool chain

3. iverilog

4. vivado gui mode

## GNU Tool chian

**Windows configuration**

1. Install **Cygwin** with packages git, python3, python2, autoconf, automake, libtool.
2. Download SiFive’s GNU Embedded Toolchain from https://www.sifive.com/software. See the ’Prebuilt RISC-V GCC Toolchain and Emulator’ section. After downloading and extracting the tarball, add the bin folder to your PATH. For Windows, make sure you can execute riscv64-unknown-elf-gcc -v in a Cygwin terminal.
3. Clone the elf2hex repo git clone git@github.com:sifive/elf2hex. Follow the instructions in the elf2hex repo README to build it from git. You should be able to run riscv64-unknown-elf-elf2hex in a terminal.（Just two python scripts）

**Tools description**

**gcc**

-mabi=ilp32 # Specify integer and floating-point calling convention. Meaning int, long and pointer are 32bit

-march=rv32i # selects the architecture to target. This controls which instructions and registers are available for the compiler to use.

-static # On systems that support dynamic linking, this overrides -pie and prevents linking with the shared libraries. On other systems, this option has no effect.

-mcmodel=medany # Generate code for the medium-any code model. The program and its statically defined symbols must be within any single 2 GiB address range.

-nostdlib # Do not use the standard system startup files or libraries(newlib) when linking

-nostartfiles # Do not use the standard system startup files when linking. The standard system libraries are used normally, unless -nostdlib, -nolibc, or -nodefaultlibs is used.

-T # Use script as the linker script

-I dir # Add the directory dir to the head of the list of directories to be searched for header files.

-Wl,-Map=map\_file.map #pass option to linker.

**objdump** #objdump displays information about one or more object files.反汇编

Example: objdump -D -Mnumeric $@ > $(basename $@).dump

-D # disassemble the contents of all sections

-Mnumeric # Pass target specific information to the disassembler.

**stripe #**program removes information from executable binary programs and object files that is not essential or required for normal and correct execution

**objcopy**

# objcopy can be used to generate a raw binary file by using an output target of ‘binary’ (e.g., use -O binary). When objcopy generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file. All symbols and relocation information will be discarded. The memory dump will start at the load address of the lowest section copied into the output file.

# conversion of the ELF into a raw binary image, strip .comment & .riscv.. sections.

# $(OBJCOPY) -O binary $(PROJ\_NAME).elf $(PROJ\_NAME).bin

**What is newlib?**

Newlib is a C library intended for use on embedded systems.(Already integrated with the compiler?)

You need to implement the interface function to use printf, malloc, etc.

**Link script**

Some GCC attributes can be used to communicate to the linker about:

* Weak symbols, which can be overridden by symbols with the same name
* Symbols to be stored in a specific section in the ELF file, defined in the linker script
* Implicitly used symbols, which prevent the linker from discarding the symbol because it is referred to nowhere in the code

**Weak attribute**

*void \_\_attribute\_\_(weak) my\_procedure(int x) { /\* do nothing \*/ }*

**Section attribute**

*const uint8\_t*

*\_\_attribute\_\_((section(“.keys”)))*

*private\_key[KEY\_SIZE] = {0};*

In this example, the array is placed in the .keys section, which requires its own entry in the linker script as well. Both function and data can use this attribute. Use for putting data and function at fixed memory address.

**linker script**

Check these links and GNU ld doc.

* https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html\_chapter/ld\_3.html
* <https://docs.huihoo.com/redhat//rhel-4-docs/rhel-ld-en-4/simple-example.html>
* https://www.youtube.com/watch?v=B7oKdUvRhQQ&ab\_channel=FastbitEmbeddedBrainAcademy

Example: https://github.com/niekiran/baremetalembedded/blob/master/source/stm32\_ls.ld

*SECTIONS*

*{*

*. = 0x10000000; //start address*

*.text : { //output section .text*

*\* (.start) //input section .start*

*\* (.text) //input section .text*

*}*

*}*

You may use the symbols in the linker script in the c program and **Remember to add &.**

**GCC Inline asm**

asm volatile ("csrw 0x51e, %[v]" :: [v]"r"(csr\_val));

asm volatile ("nop");

**CAT multiple image**

$ arm-none-eabi-objcopy -O binary --pad-to=4096 --gap-fill=0xFF bootloader.elf bootloader.bin

$ arm-none-eabi-objcopy -O binary app.elf app.bin

$ cat bootloader.bin app.bin > image.bin

如果第二个image是从4096地址开始的所以第一个bin文件要padding到4096 Bytes.

## Hardware design tool

**Iverilog**

**-s** topmodule

**-I** include dir