National Yang Ming Chiao Tung University
Computer Architecture & System Lab

# Get Your Lab CPU to Run C on FPGA

## CS10014 Computer Organization

Tsung Tai Yeh

Department of Computer Science
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

# Don’t Panic

- This is not an assignment or lab, just a demonstration
- **You are not asked to do it**
- May be cool if you want to

# Outline

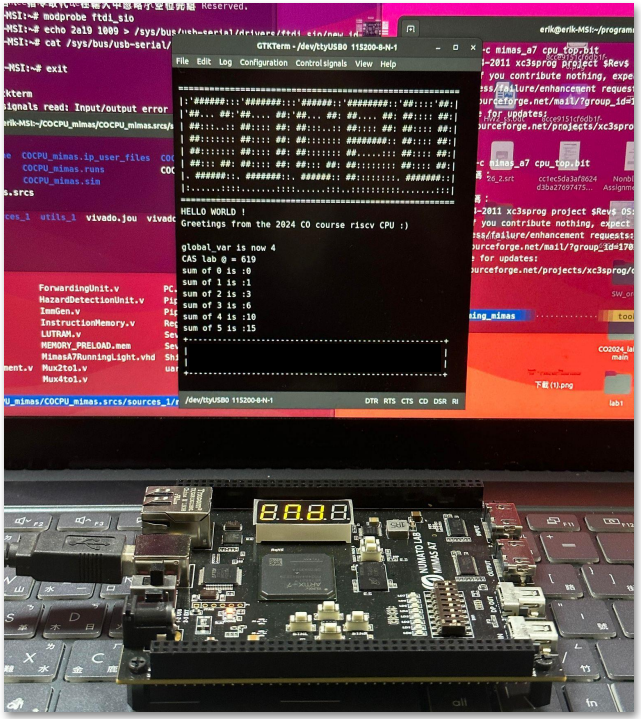
- What is this?
- The FPGA we are using
- About the whole HW-SW package
- Some additional requirements
- How to use it

# What is this?

## A project for your CPU to run some very simple C on FPGA

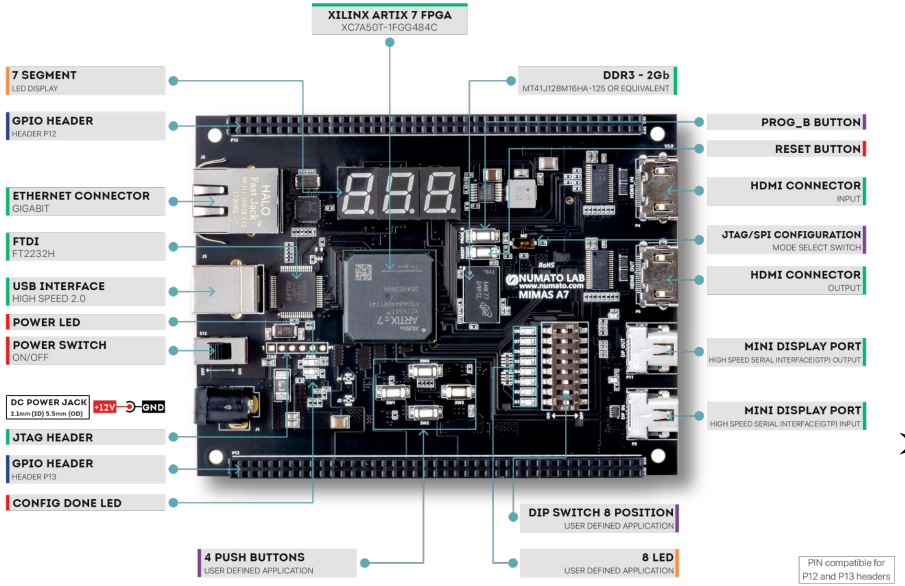
```
int global_var = 0;
int main() {
  printf("================================================ \n");
  printf("|:'#####:::'####:::'#####::'######::'##:::'##:|\n");
  printf("| ##::::.: #::: #: ##::..:: #::: ##: ##::: ##:|\n");
  printf("| ##:::::: #::: #: ##:::::: ######:: ##::: ##:|\n");
  printf("| ##:::::: #::: #: ##:::::: #....::: ##::: ##:|\n");
  printf("|. #####::. ####::. #####:: #::::::. ######::|\n");
  printf("================================================\n");
  printf("HELLO WORLD !\n");
  printf("Greetings from the 2024 CO course RISC V CPU :) \n");
  g_var += 4;
  printf(" global_var is now %d\n", g_var);
  printf(" CAS lab @ = %d\n", 600 + 10 + 9);
  printFib(7);

. . . and so on ...
```

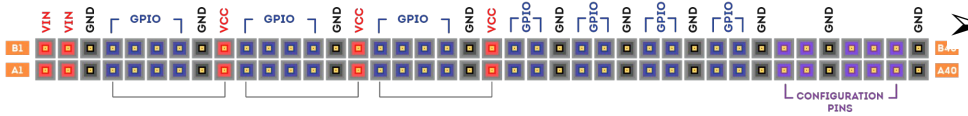
Your lab CPU On FPGA →




< Just an example, of course you may write yours

# The FPGA we are using — **Numato Lab Mimas A7**




! **Requires software from Numato Lab when writing bitstream to FPGA**
- Unless the system is powered by external DC, then the XVC server is available for use.

! **Requires manually adding VID:PID to FTDI driver when using Linux**

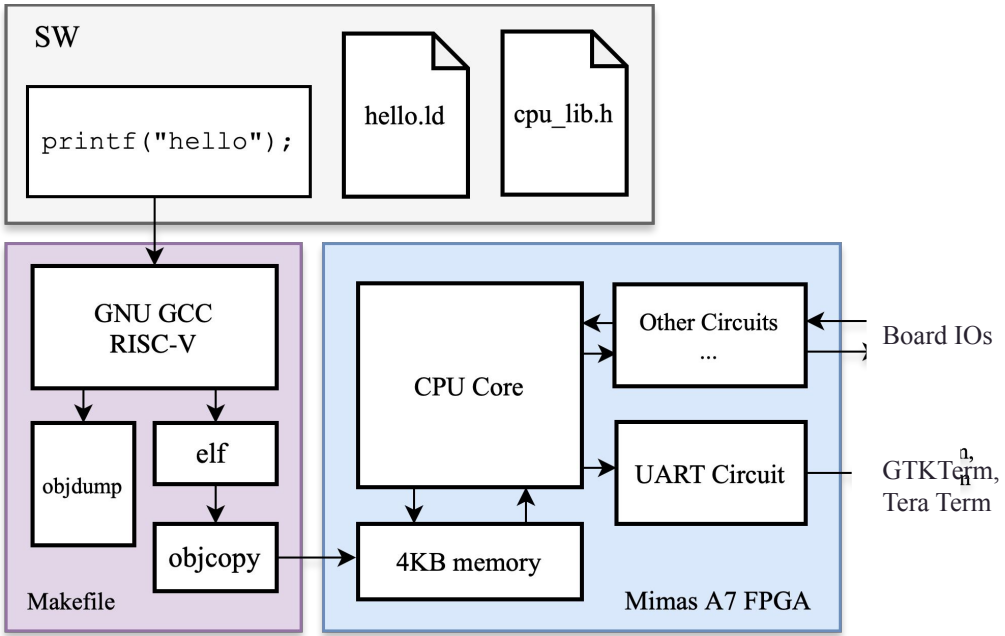
➢ Uses multiplexing to display the 4 digits
- Our recommend refresh rate for the 4 digits is 320 Hz.

➢ UART through the usb-uart bridge
- Pin Y22 for rx and Y21 for tx.

# System Diagram

Contains the following two parts:

- Software
  - Demo library for CPU
  - Linker script
  - Makefile
- Hardware
  - D-mem & I-mem using LUTRAM
  - Vivado project for Mimas A7
  - Top module w/ UART† and stuffs
  - Board constraints

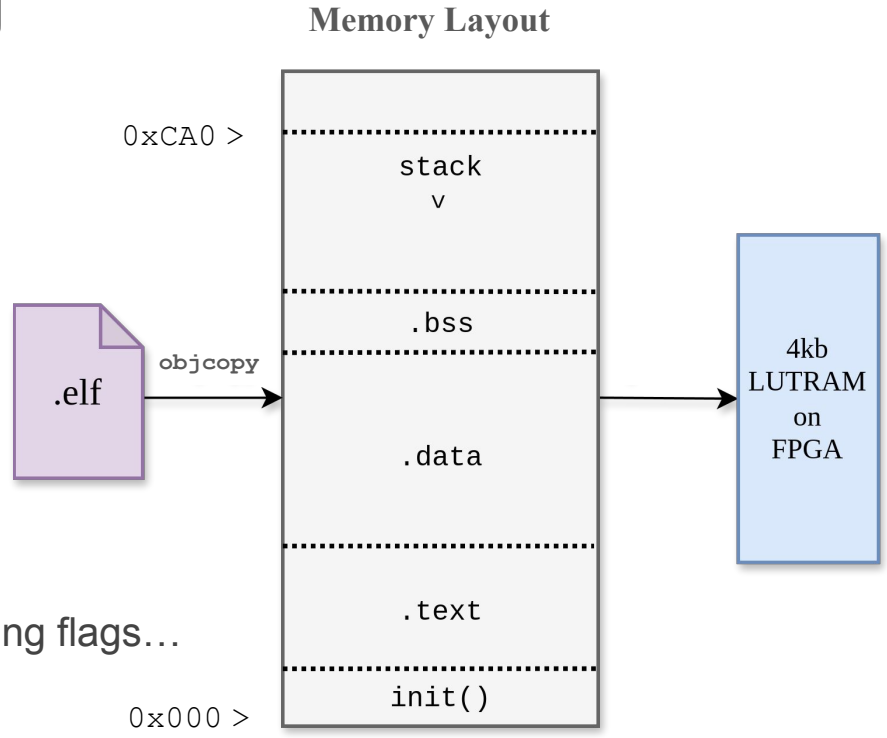



† UART circuit from https://nandland.com/uart-serial-port-module/

# Software of the Lab CPU

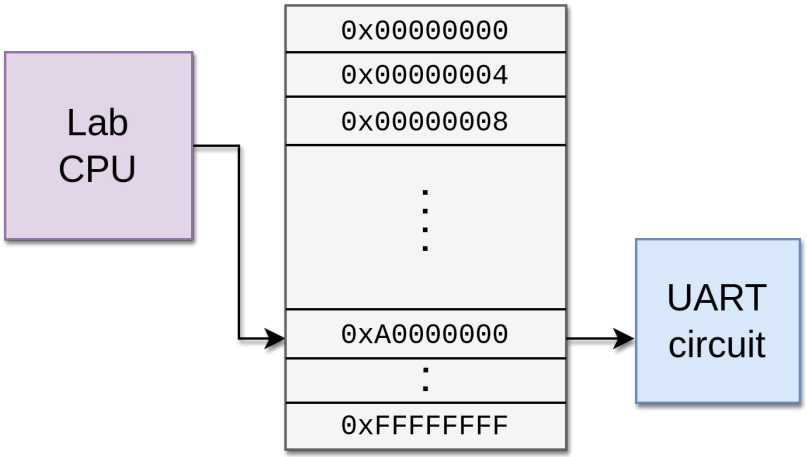
The software package contains:

- Minimal library for CPU
  - printf() implementation for the system
  - init() as init function
  - exit() after main
- Linker script
  - Memory layout
- Makefile
  - Integrates xxd, objdump, objcopy, compiling flags…

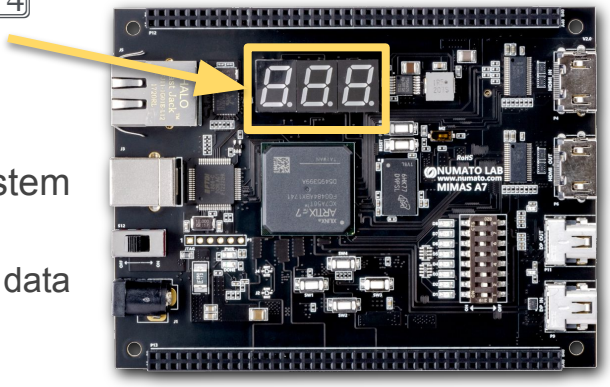


# How printf() in this system works

- ASCIIs will be sent to UART circuit on top module by ‘sw’ instruction
  - Write some data to pointer that points to some specific address, then send to uart circuit



Actually, there are four 4



# 7-Segment Self-Check Codes

- The 7-seg display provides some simple information about your system
  - We test some basic “sw” instructions before entering the main function
  - CPU writes some data to the self-check circuit, and check if sw addr == data

`t01`

No data written to the self-check circuit, “sw” or others may fail.

`E01`

Data written to the self-check circuit is wrong (addr != data), check the datapath of writing data memory.

`run`

Simple sw test passed, trigger “run” before entering main.

`End`

Returned from main, execution ends.

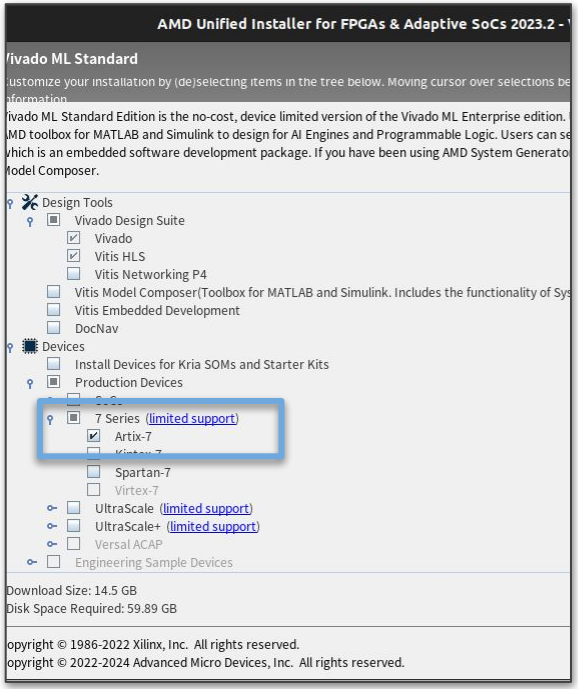
# Some additional requirements

➢ In addition to the lab 4 instruction requirements, there are still some other instructions that are required to implement for running the demo C program.

- **`LBU [Load Byte Unsigned]`**
  - > 'putc' in 'printf' sends one char at a time to the UART circuit.
- **`LUI [Load Upper Immediate]`**
  - > For CPU to load some numbers larger than 4096.
    Ex. `for(int i = 0; i < 4000; i++)` , compiler actually 'lui' loads 4096 then -96 to get 4000.
    (riscv32-unknown-elf-gcc -O2)
- **`SLL [Shift Left Logical]`**
  - > 1 << n is same as $2^n$ but faster.

# About Vivado

1. Install your Vivado with package for Artix-7




2. Download the BSP file for Mimas A7 from the MimasA7 directory

```
https://github.com/numato/vivadoBSP
```

3. Place it under:

/path/to/Vivado/2023.2/data/xhub/boards/XilinxBoardStore/boards

```
+---- boards
|    Xilinx
|    MimasA7 << whole directory here
|
```

# Building the Toolchain

Since most of us don't use a RISC-V machine, we need a cross compile environment to generate RISC-V asm.

1. Clone the RISC-V GNU compiler from github.

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain
```

*The whole repo is around **6.5GB**, so cloning it may takes plenty of time!*

2. Compile only the riscv32-unknown-elf-gcc toolchain with parameter rv32i, since our lab CPU supports integer only.

```
./configure --prefix=/opt/riscv --disable-linux --with-arch=rv32i
make -j8
```

*Also takes plenty of time.*

# Writing some C for your CPU

Under the SW_dev directory, there are the following files:

```
+---- SW_dev
|    cpu_lib.h
|    cpu_lib.c
|    hello.c
|    hello.ld
|    Makefile
```

`cpu_lib.c:`

Some minimal functions and parameters for the system to work.

`hello.c:`

Write your hello world here.

`hello.ld:`

Linker script to perform the specific layout for our CPU.

`Makefile:`

This makefile will export the PATH and env variables automatically.

Modify the makefile if your prefix at toolchain compiling is not default.

➢ Compile your program with:

```
$ make
```

# Writing some code for your CPU (Cont.)

After ‘make’, some files are generated…

- `hello.objdump`

```
000003e4 <main>:
 3e4:    fe010113            add   sp,sp,-32
 3e8:    01312623            sw    s3,12(sp)
 3ec:    57800513            li    a0,1400
 3f0:    00112e23            sw    ra,28(sp)
                                  And so on
```

You may check if there are unsupported instructions generated.

Beware that some of them are pseudo-instructions.

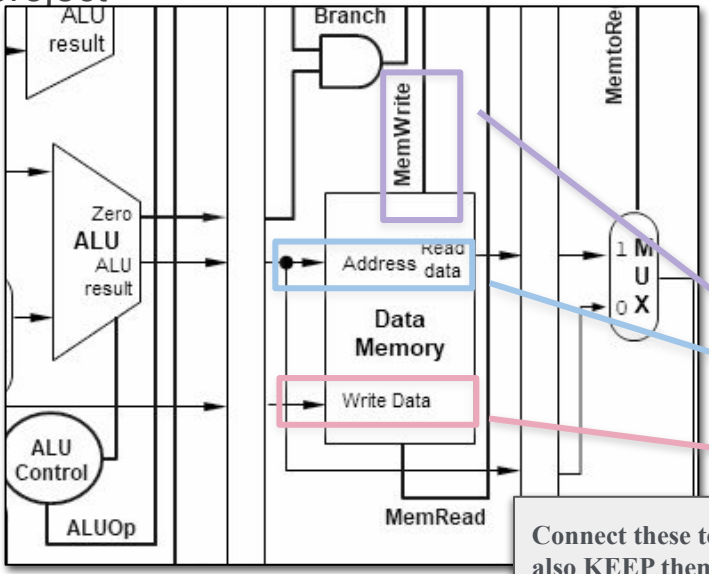
- `mem_preload.mem`

```
00000011001001110000000001010100
00000011001001101100000001010011
10000011001001101000000001010011
00010011000000010000000111111110
```

32-bit binaries just like the previous lab.

# Connecting to the Top Module

Modify the IO ports of your CPU to connect to the top module of this project




(Module definition from previous labs)

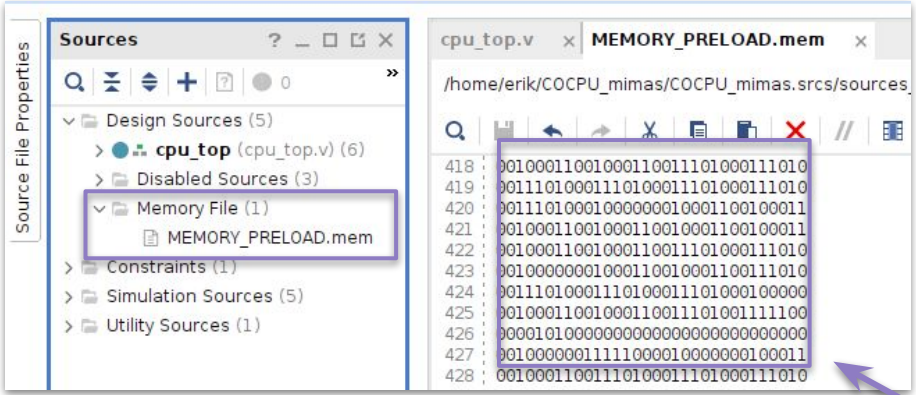
```
module PipelineCPU
(
    input clk,
    input start,
    output [31:0] r [0:31]
);
```

**Self-check code "t01" & "E01" will be triggered if you don't connect them correctly!**

```
module PipelineCPU
(
    input          clk,
    input          start,
    output [31:0]  cp0_mem_addr
    output [31:0]  cp0_wr_data
    output         cp0_wr_enabl
);
```

**Connect these to the IO port, <u>also KEEP them connected to DATA MEMORY as usual!</u>**

# Load the compiled binaries to your CPU




Load the compiled binary file to the MEMORY_PRELOAD.mem in the vivado project.

Just like loading the instructions in previous lab.

- `mem_preload.mem`

```
00000011001001110000000001010100
00000011001001101100000001010011
10000011001001101000000001010011
00010011000000010000000111111110
```

Then generate bitstream

# Programming the Mimas A7 (Debian based Linux)

The Mimas A7 does not work well with Vivado without DC power supply or JTAG currently (especially Linux), so we depend on some third-party tools to program our FPGA.

1. Install XC3SProg.

```
$ sudo apt-get install xc3sprog
```

2. Download the Mimas A7 XC3SProg dependency files for linux here.

```
numato.com/download/xc3sprog-dependency-files-for-linux/
```

3. Unzip all the files to a new directory, and cd to that directory, then use the following command to program the Mimas A7 FPGA.

```
$ xc3sprog -c mimas_a7 <bitstream_file_path>
```

Find your bitstream file at `/path/to/COCPU_mimas/COCPU_mimas.runs/impl_1/cpu_top.bit`

# Programming the Mimas A7 (Windows)

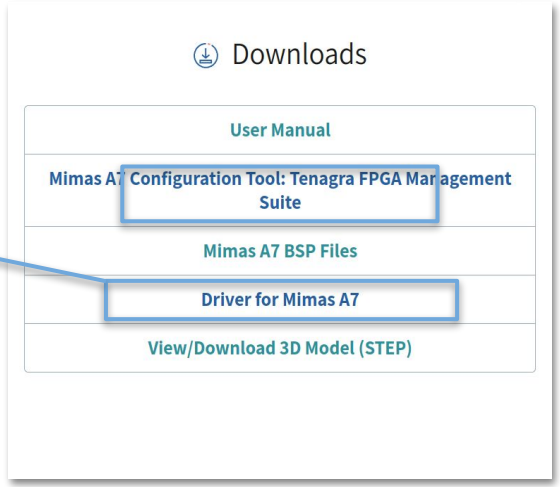
Compared to poorly supported Linux, we have some official tools for windows, if the FPGA is powered with DC, you may use XVC server to program directly from Vivado.

Since we are not providing the DC power supply, there are some instructions on using the official ‘Tenagra’ software to program it.

1. Go to the Mimas A7 product page

```
https://numato.com/product/mimas-a7-artix-7-fpga-development-board/
```

2. Download ‘Tenagra’ and ‘Driver for Mimas A7’.

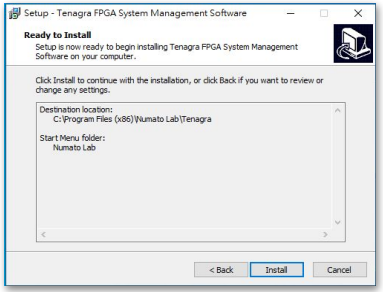



```
// BEFORE YOU DOWNLOAD THEM ...
Due to our experiences, the stability of Tenagra is pretty
poor.
Native Ubuntu is our most recommended development environment.
```

# Programming the Mimas A7 (Windows) (Cont.)

3. Install the driver using administrator


4. Install Tenagra




5. Make sure the drivers are ready to use


6. Tenagra shall identify Mimas automatically when plugged in, click select to program the .bin


! Find your bitstream file at
/path/to/COCPU_mimas/COCPU_mimas.runs/impl_1/cpu_top.bit

# USB-UART bridge

## Linux

To get your computer to display the ASCII that your CPU sent, there is one more step left.

```
$ sudo modprobe ftdi_sio
$ echo 2a19 1009 > /sys/bus/usb-serial/drivers/ftdi_sio/new_id
$ cat /sys/bus/usb-serial/drivers/ftdi_sio/new_id
```

*Check if the VID / PID is written correctly.*

Now you may find the USB port in the GTKTerm or on the tty device list, set the baud rate to 230400 (we recommend the GTKTerm).

## Windows

Simply choose the MimasA7 COM port and set the baud rate to 230400 (Tera Term is recommended).