RISC-V TRAINING MATERIAL

Imagination University Programme (IUP)

RVfpga (RISC-V FPGA) 교육자료

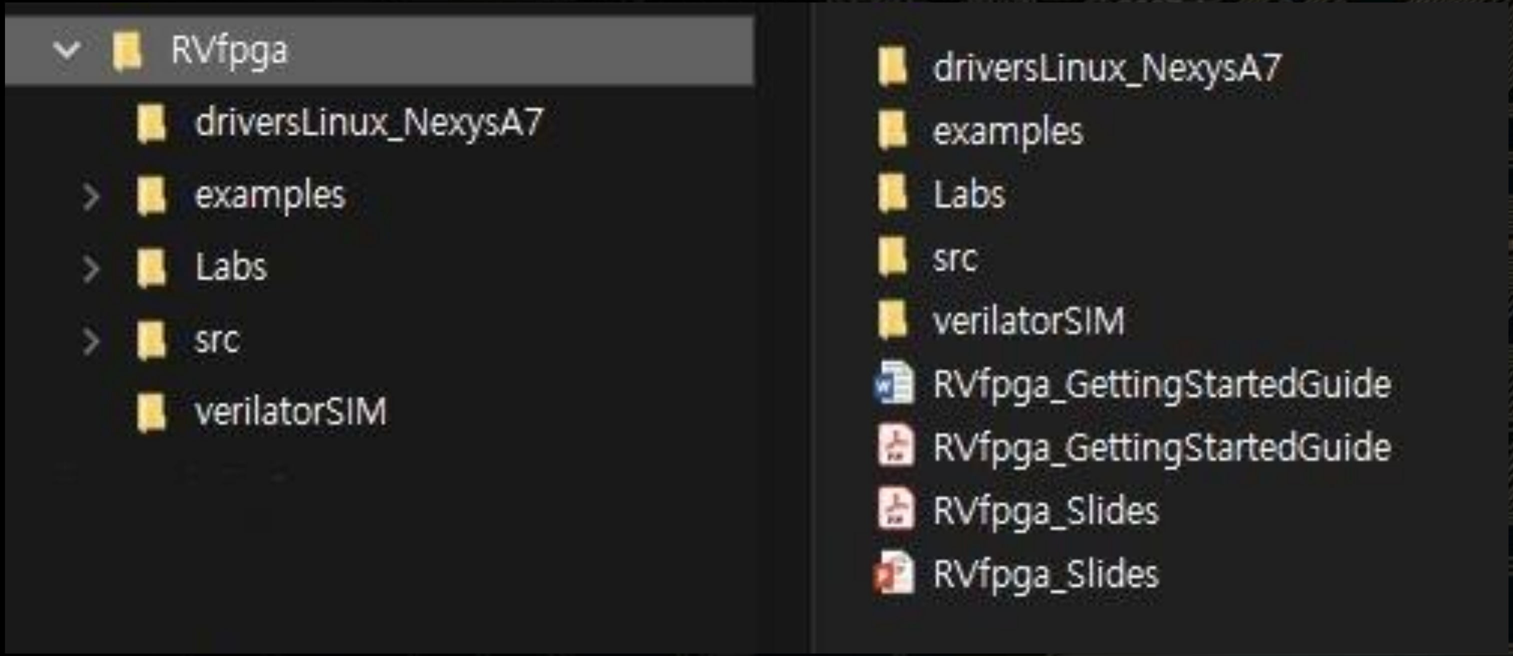
- 명령어, 개발 도구, 예제/실습 자료
- 상용 RISC-V chip의 FPGA 적용
- 상용 SoC 설계를 위한 자료
- 컴퓨터 구조, 디지털 디자인, 임베디드 시스템, 프로그래밍 교육자료
- 기술 교육 강의하시는 분을 위한 교육 자료

교육 자료 구성

- RISC-V 구조 / RVfpga 설명
- RVfpga 하드웨어 사용
- RVfpga 시뮬레이션
- 예제

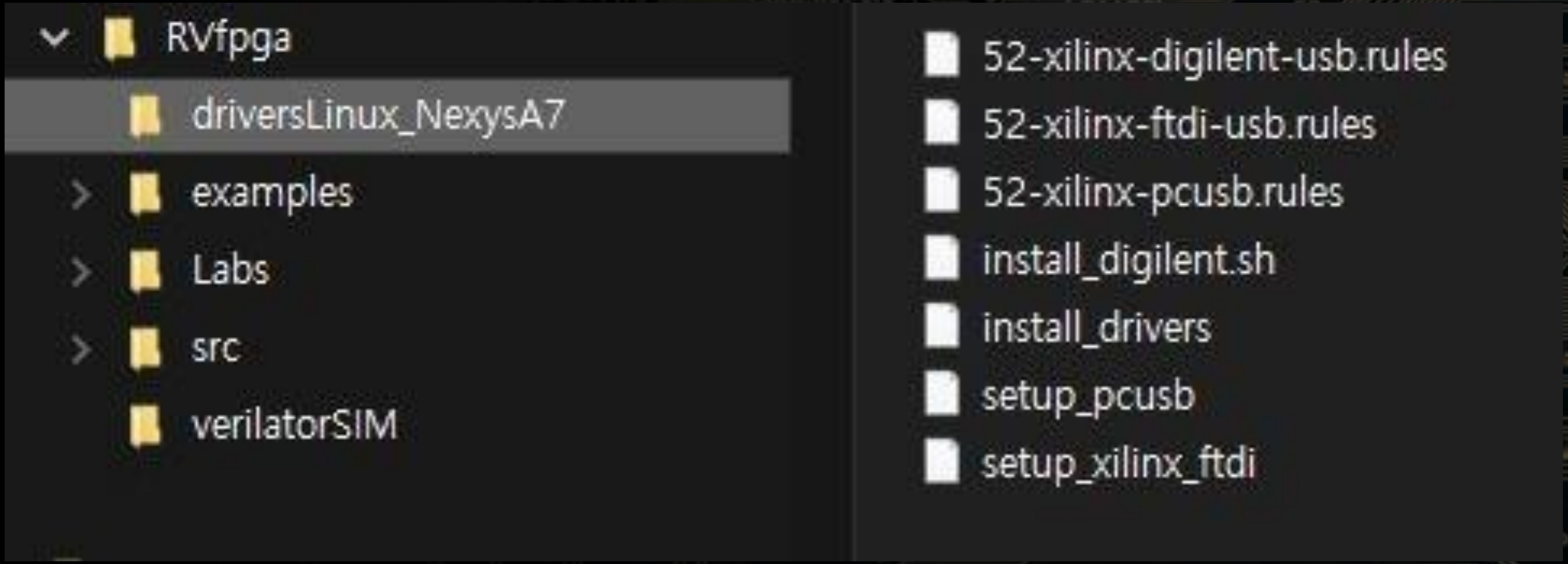
  OpenOCD, PlatformIO, Verilator, GTKWave
  Vivado, IoT application
  Nexys A7 FPGA Board
  Western Digital SweRV EH1 Core

교육 자료 파일 1



RVfpga Home Folder

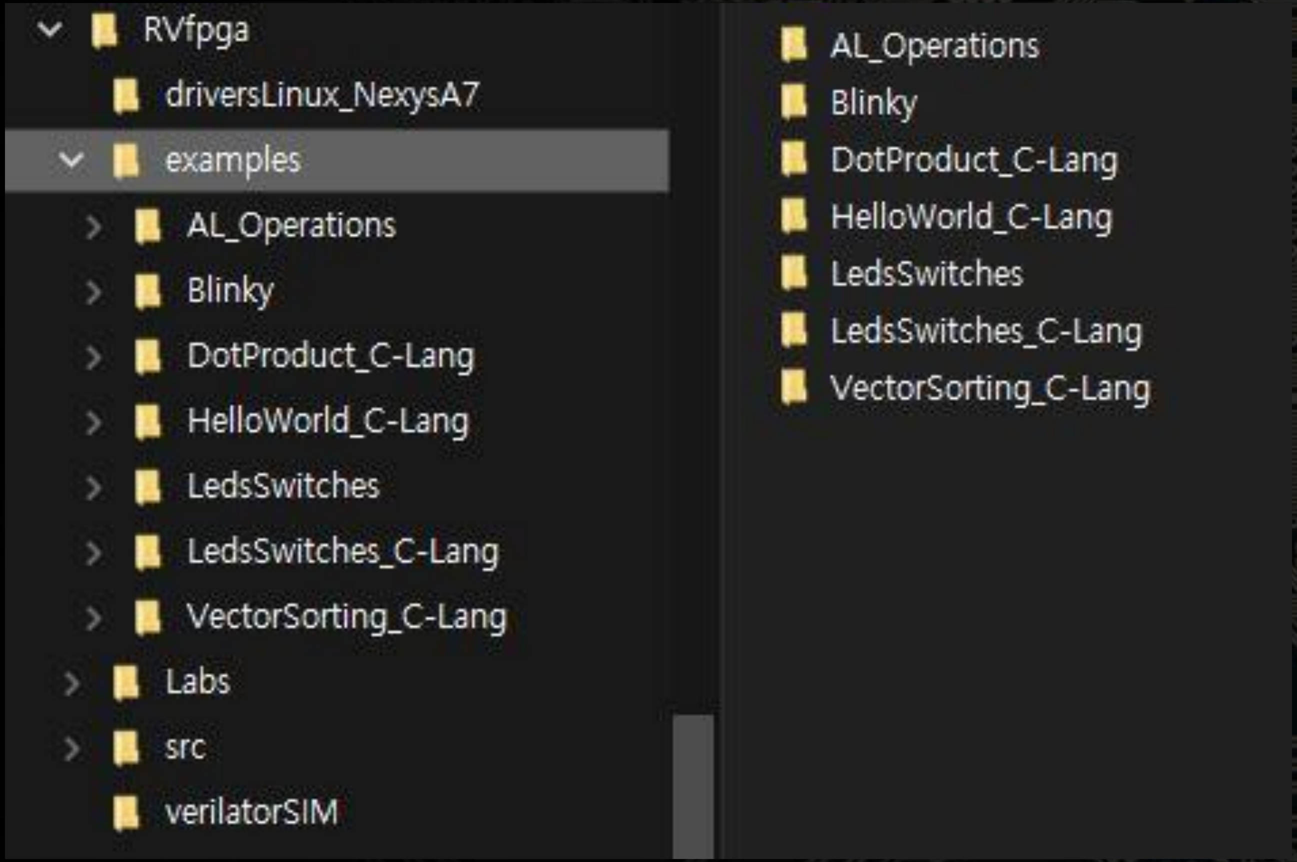
교육 자료 파일 2



52-xilinx-digilent-usb.rules
52-xilinx-ftdi-usb.rules
52-xilinx-pcusb.rules
install_digilent.sh
install_drivers
setup_pcusb
setup_xilinx_ftdi

Linux Drivers of Nexys A7 Board

교육 자료 파일 3

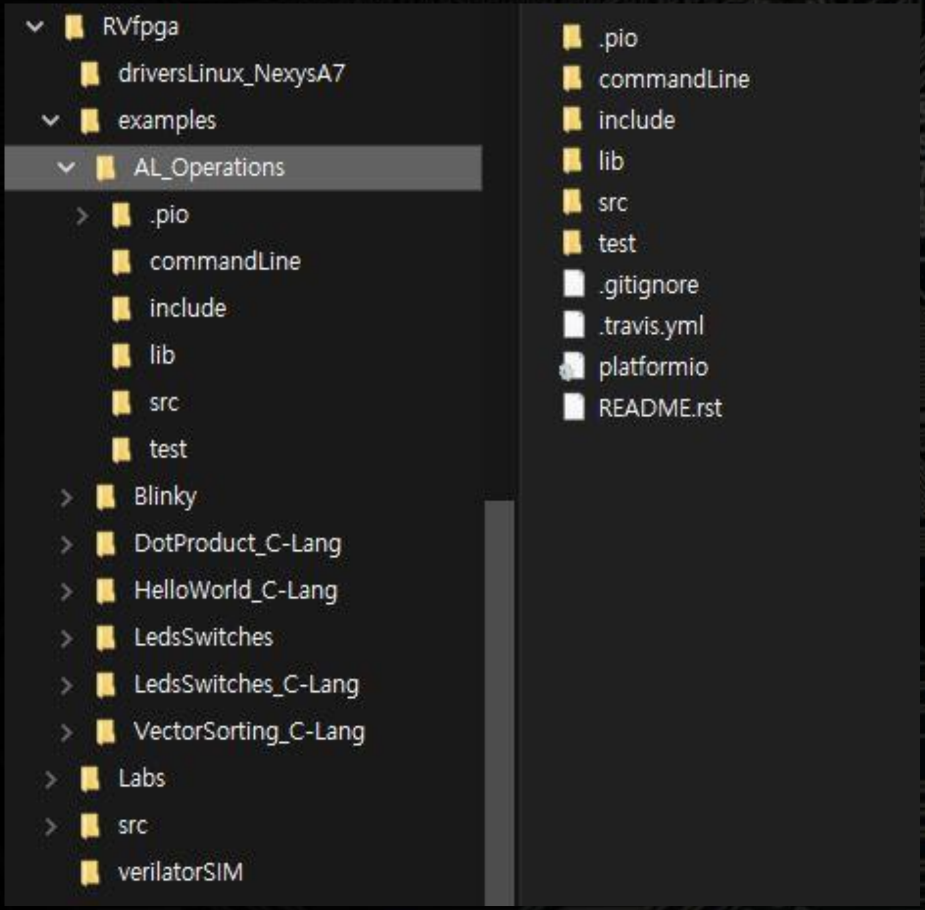


```
RVfpga
    driversLinux_NexysA7
    examples
        AL_Operations
        Blinky
        DotProduct_C-Lang
        HelloWorld_C-Lang
        LedsSwitches
        LedsSwitches_C-Lang
        VectorSorting_C-Lang
    Labs
    src
    verilatorSIM
```

```
AL_Operations
Blinky
DotProduct_C-Lang
HelloWorld_C-Lang
LedsSwitches
LedsSwitches_C-Lang
VectorSorting_C-Lang
```

실습용 예제 파일

교육 자료 파일 4

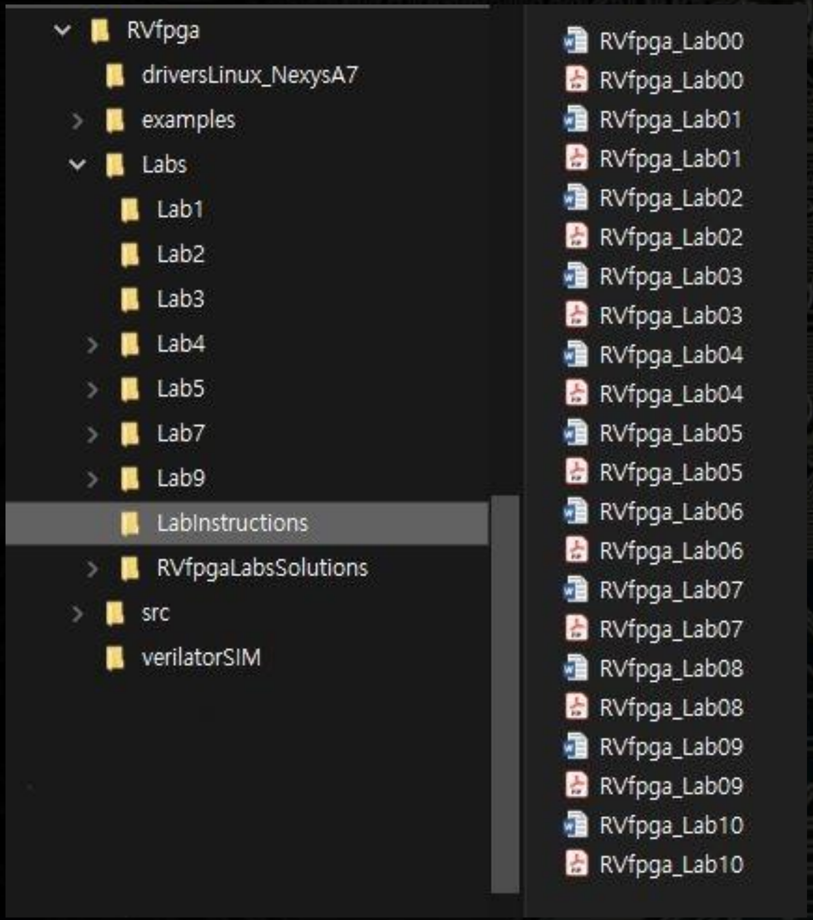


```
RVfpga
  driversLinux_NexysA7
  examples
    AL_Operations
      .pio
      commandLine
      include
      lib
      src
      test
    Blinky
    DotProduct_C-Lang
    HelloWorld_C-Lang
    LedsSwitches
    LedsSwitches_C-Lang
    VectorSorting_C-Lang
  Labs
  src
  verilatorSIM
```

```
.pio
commandLine
include
lib
src
test
.gitignore
.travis.yml
platformio
README.rst
```

실습용 예제 파일

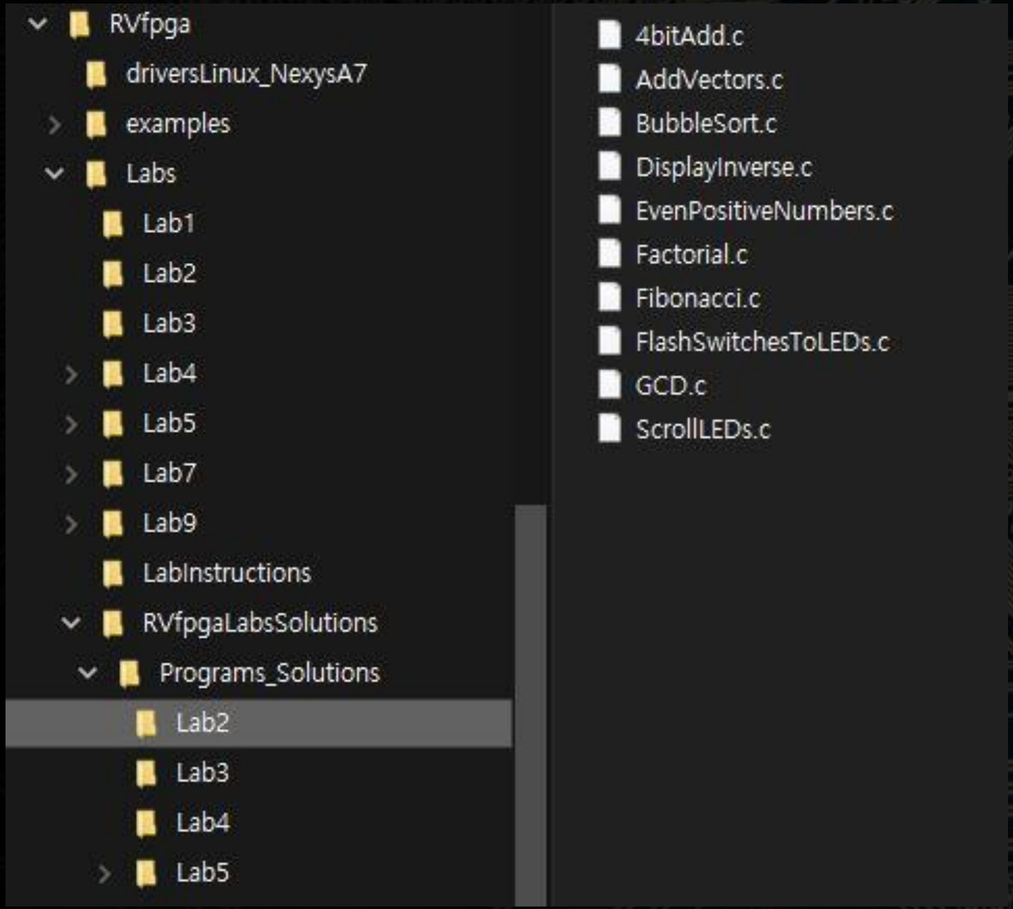
교육 자료 파일 5


RVfpga
driversLinux_NexysA7
examples
Labs
Lab1
Lab2
Lab3
Lab4
Lab5
Lab7
Lab9
LabInstructions
RVfpgaLabsSolutions
src
verilatorSIM

RVfpga_Lab00
RVfpga_Lab00
RVfpga_Lab01
RVfpga_Lab01
RVfpga_Lab02
RVfpga_Lab02
RVfpga_Lab03
RVfpga_Lab03
RVfpga_Lab04
RVfpga_Lab04
RVfpga_Lab05
RVfpga_Lab05
RVfpga_Lab06
RVfpga_Lab06
RVfpga_Lab07
RVfpga_Lab07
RVfpga_Lab08
RVfpga_Lab08
RVfpga_Lab09
RVfpga_Lab09
RVfpga_Lab10
RVfpga_Lab10

실습 강사용 해설 및 교재

교육 자료 파일 6

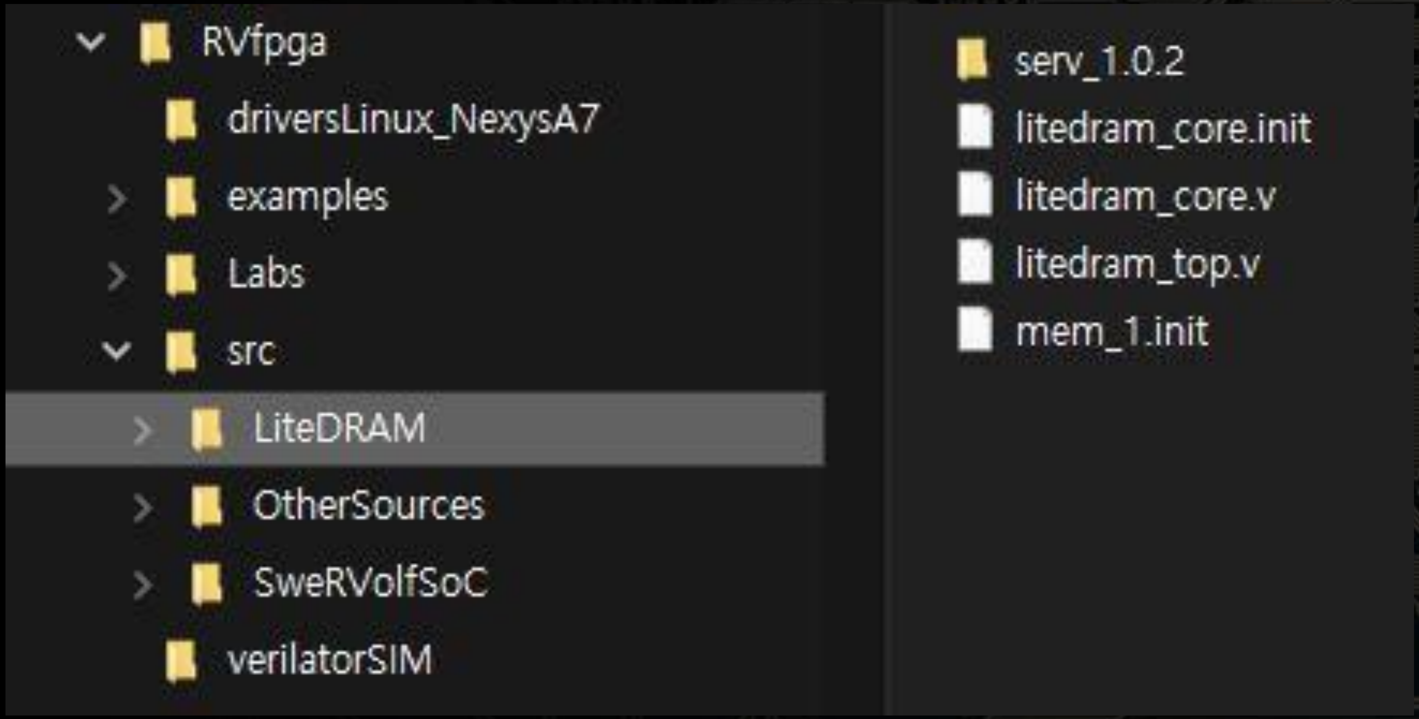


```
RVfpga
  driversLinux_NexysA7
  examples
  Labs
    Lab1
    Lab2
    Lab3
    Lab4
    Lab5
    Lab7
    Lab9
    LabInstructions
    RVfpgaLabsSolutions
      Programs_Solutions
        Lab2
        Lab3
        Lab4
        Lab5
```

```
4bitAdd.c
AddVectors.c
BubbleSort.c
DisplayInverse.c
EvenPositiveNumbers.c
Factorial.c
Fibonacci.c
FlashSwitchesToLEDs.c
GCD.c
ScrollLEDs.c
```

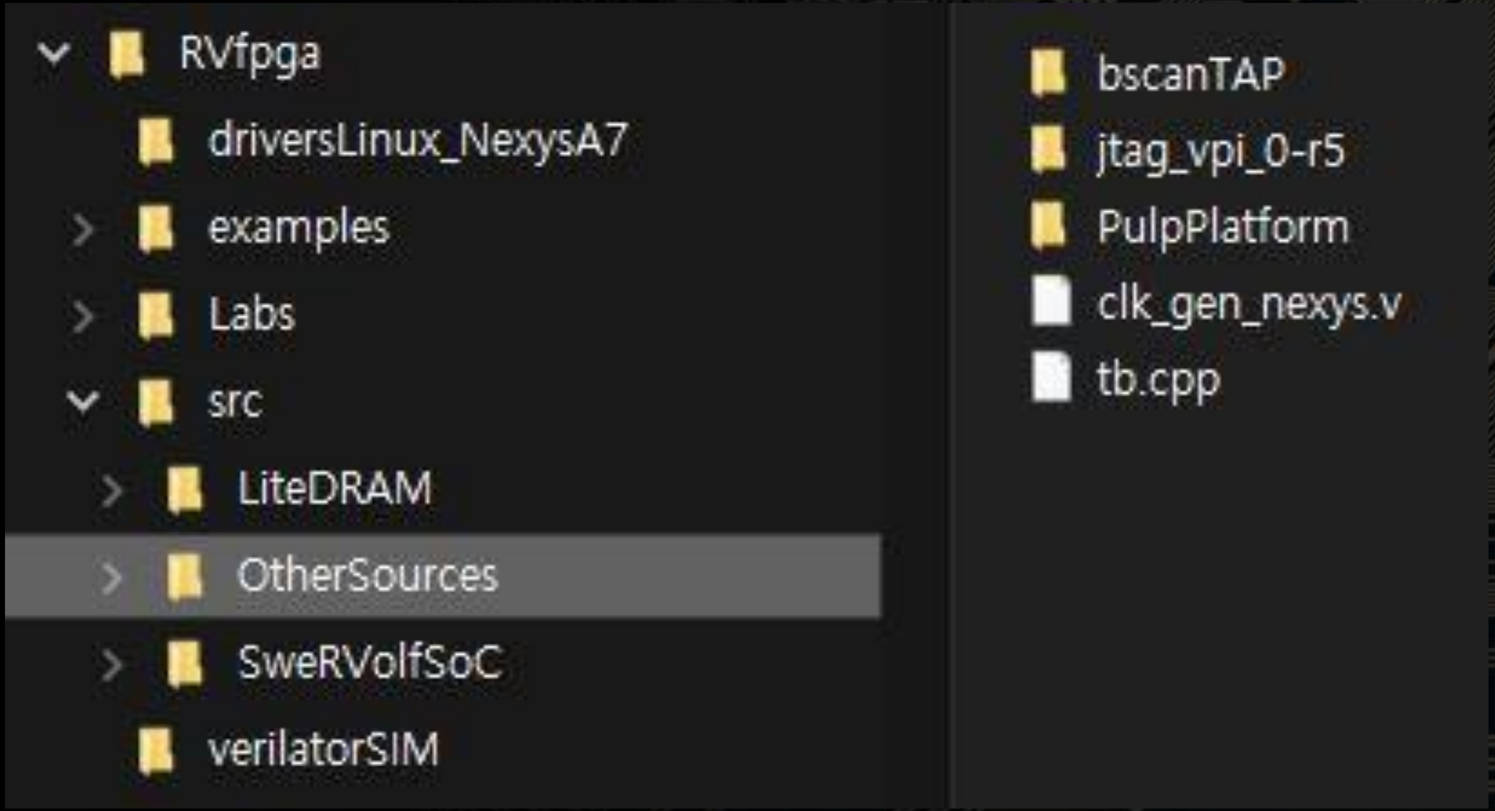
실습 예제, 응용 프로그램

교육 자료 파일 7



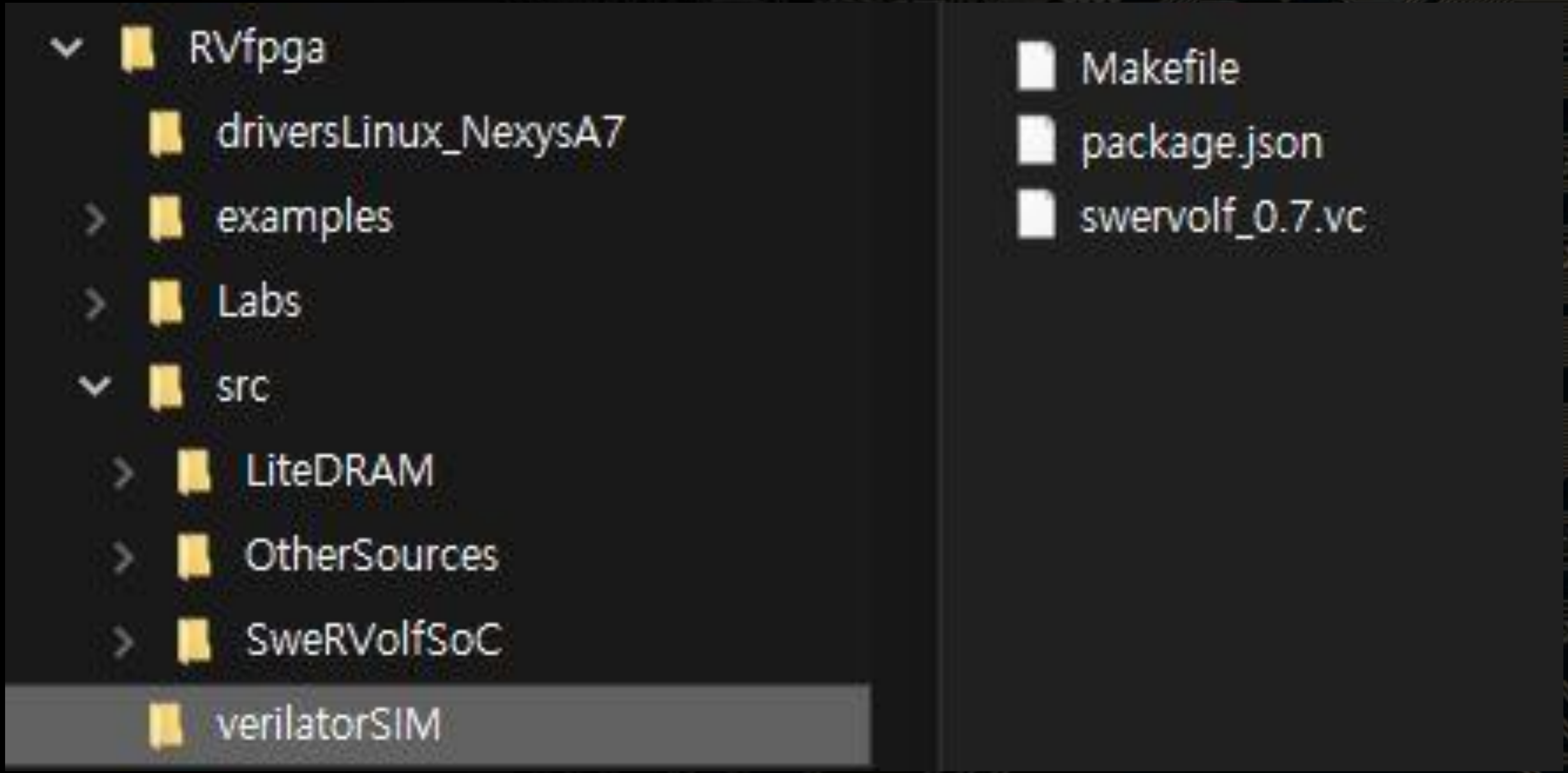
실습 예제, Lite DRAM

교육 자료 파일 8



실습 예제, Boundary Scan / JTAG

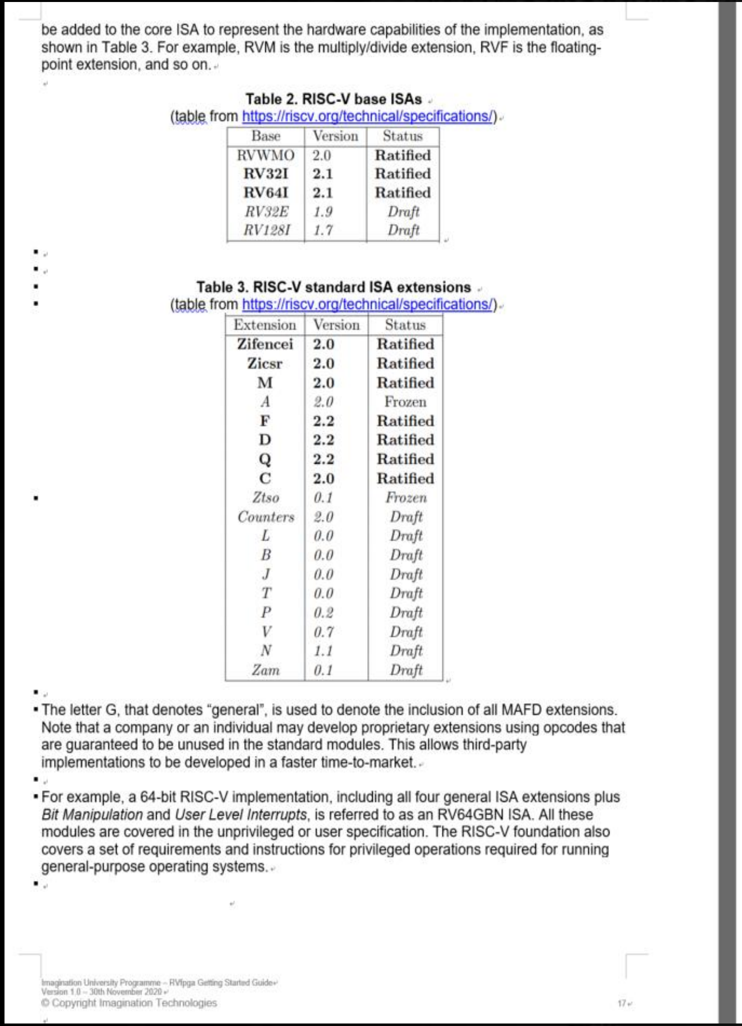
교육 자료 파일 9


RVfpga
driversLinux_NexysA7
examples
Labs
src
LiteDRAM
OtherSources
SweRVolfSoC
verilatorSIM

Makefile
package.json
swervolf_0.7.vc

실습 예제, Verilator Simultion

교육 자료 파일 10



be added to the core ISA to represent the hardware capabilities of the implementation, as shown in Table 3. For example, RVM is the multiply/divide extension, RVF is the floating-point extension, and so on.

Table 2. RISC-V base ISAs
(table from https://riscv.org/technical/specifications/)

Base	Version	Status
RVWMO	2.0	Ratified
RV32I	**2.1**	**Ratified**
RV64I	**2.1**	**Ratified**
RV32E	*1.9*	*Draft*
RV128I	*1.7*	*Draft*

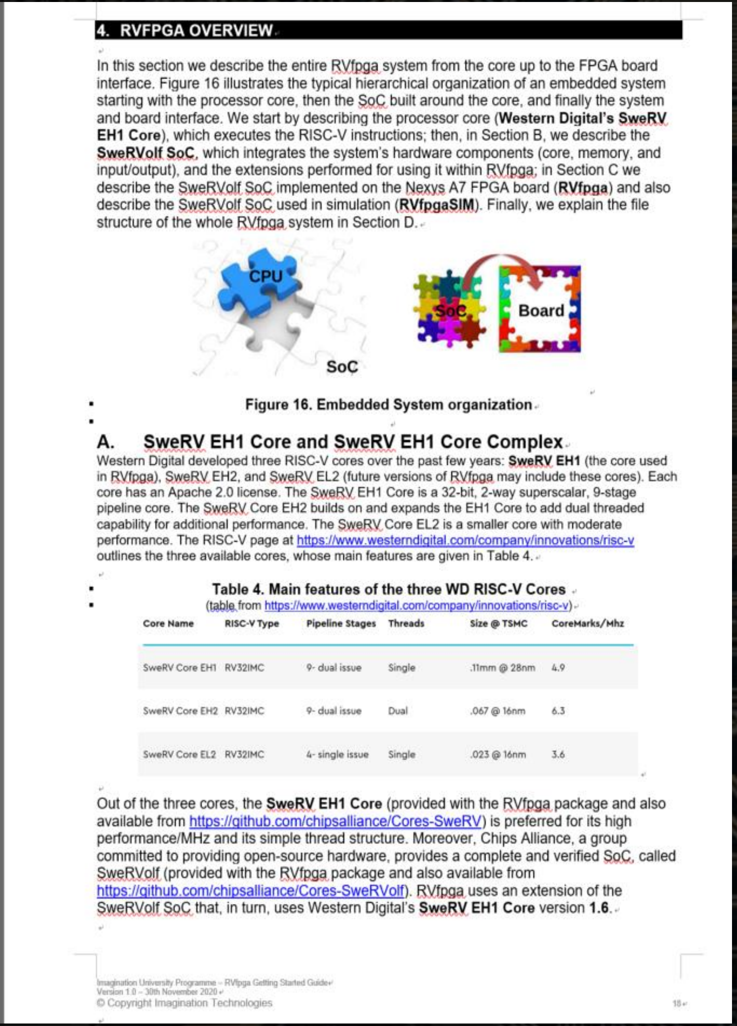
Table 3. RISC-V standard ISA extensions
(table from https://riscv.org/technical/specifications/)

Extension	Version	Status
Zifencei	**2.0**	**Ratified**
Zicsr	**2.0**	**Ratified**
M	**2.0**	**Ratified**
A	*2.0*	Frozen
F	**2.2**	**Ratified**
D	**2.2**	**Ratified**
Q	**2.2**	**Ratified**
C	**2.0**	**Ratified**
Ztso	*0.1*	*Frozen*
Counters	*2.0*	*Draft*
L	*0.0*	*Draft*
B	*0.0*	*Draft*
J	*0.0*	*Draft*
T	*0.0*	*Draft*
P	*0.2*	*Draft*
V	*0.7*	*Draft*
N	*1.1*	*Draft*
Zam	*0.1*	*Draft*

The letter G, that denotes "general", is used to denote the inclusion of all MAFD extensions. Note that a company or an individual may develop proprietary extensions using opcodes that are guaranteed to be unused in the standard modules. This allows third-party implementations to be developed in a faster time-to-market.

For example, a 64-bit RISC-V implementation, including all four general ISA extensions plus *Bit Manipulation* and *User Level Interrupts*, is referred to as an RV64GBN ISA. All these modules are covered in the unprivileged or user specification. The RISC-V foundation also covers a set of requirements and instructions for privileged operations required for running general-purpose operating systems.

Imagination University Programme – RVfpga Getting Started Guide
Version 1.0 – 30th November 2020
© Copyright Imagination Technologies 17



4. RVFPGA OVERVIEW

In this section we describe the entire RVfpga system from the core up to the FPGA board interface. Figure 16 illustrates the typical hierarchical organization of an embedded system starting with the processor core, then the SoC built around the core, and finally the system and board interface. We start by describing the processor core (**Western Digital's SweRV EH1 Core**), which executes the RISC-V instructions; then, in Section B, we describe the **SweRVolf SoC**, which integrates the system's hardware components (core, memory, and input/output), and the extensions performed for using it within RVfpga; in Section C we describe the SweRVolf SoC implemented on the Nexys A7 FPGA board (**RVfpga**) and also describe the SweRVolf SoC used in simulation (**RVfpgaSIM**). Finally, we explain the file structure of the whole RVfpga system in Section D.

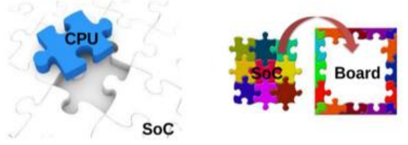


Figure 16. Embedded System organization

A. SweRV EH1 Core and SweRV EH1 Core Complex

Western Digital developed three RISC-V cores over the past few years: **SweRV EH1** (the core used in RVfpga), SweRV EH2, and SweRV EL2 (future versions of RVfpga may include these cores). Each core has an Apache 2.0 license. The SweRV EH1 Core is a 32-bit, 2-way superscalar, 9-stage pipeline core. The SweRV Core EH2 builds on and expands the EH1 Core to add dual threaded capability for additional performance. The SweRV Core EL2 is a smaller core with moderate performance. The RISC-V page at https://www.westerndigital.com/company/innovations/risc-v outlines the three available cores, whose main features are given in Table 4.

Table 4. Main features of the three WD RISC-V Cores
(table from https://www.westerndigital.com/company/innovations/risc-v)

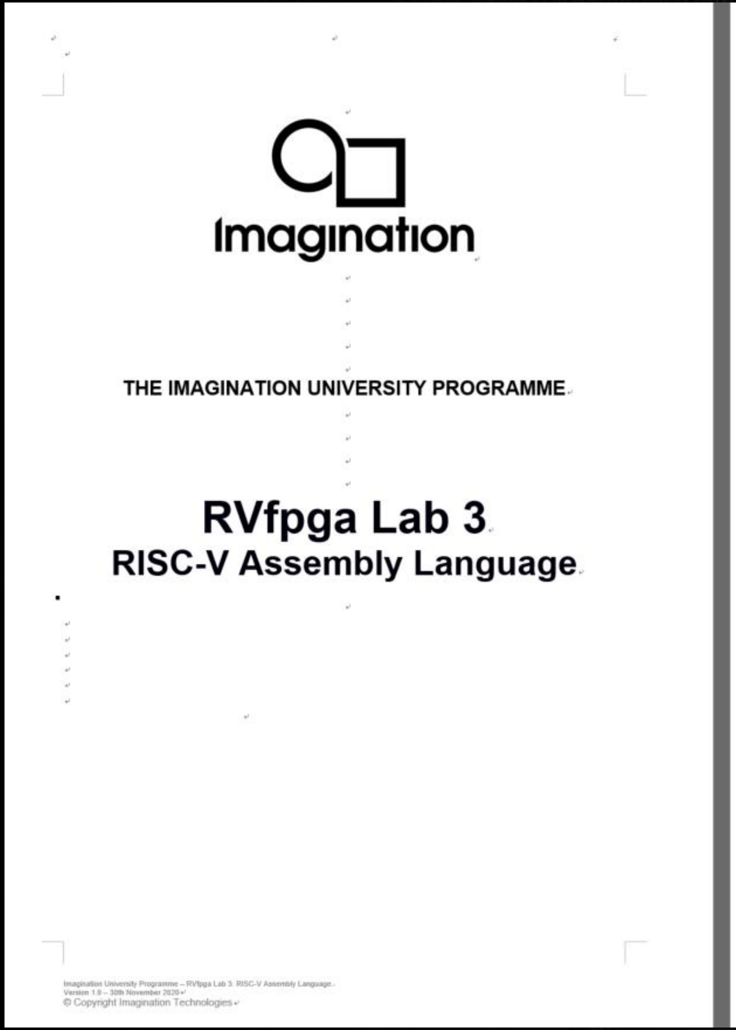
Core Name	RISC-V Type	Pipeline Stages	Threads	Size @ TSMC	CoreMarks/Mhz
SweRV Core EH1	RV32IMC	9- dual issue	Single	.11mm @ 28nm	4.9
SweRV Core EH2	RV32IMC	9- dual issue	Dual	.067 @ 16nm	6.3
SweRV Core EL2	RV32IMC	4- single issue	Single	.023 @ 16nm	3.6

Out of the three cores, the **SweRV EH1 Core** (provided with the RVfpga package and also available from https://github.com/chipsalliance/Cores-SweRV) is preferred for its high performance/MHz and its simple thread structure. Moreover, Chips Alliance, a group committed to providing open-source hardware, provides a complete and verified SoC, called SweRVolf (provided with the RVfpga package and also available from https://github.com/chipsalliance/Cores-SweRVolf). RVfpga uses an extension of the SweRVolf SoC that, in turn, uses Western Digital's **SweRV EH1 Core** version **1.6**.

Imagination University Programme – RVfpga Getting Started Guide
Version 1.0 – 30th November 2020
© Copyright Imagination Technologies 18

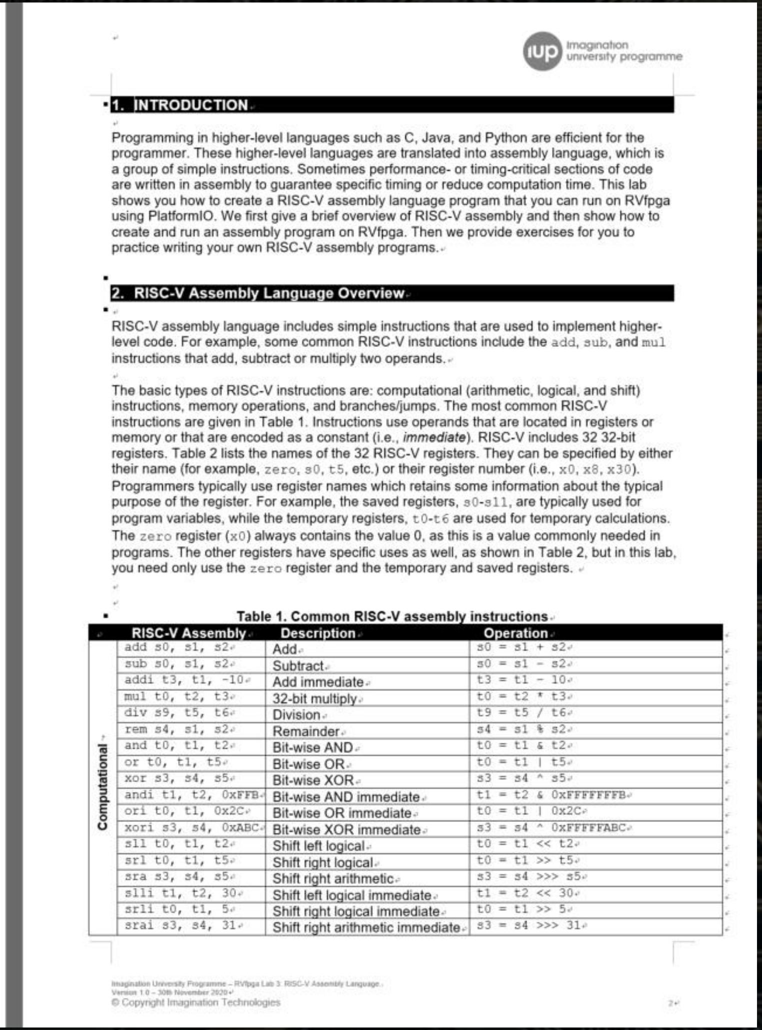
RVfpga
가이드, 102 pages

교육 자료 파일 11



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 3
RISC-V Assembly Language



1. INTRODUCTION

Programming in higher-level languages such as C, Java, and Python are efficient for the programmer. These higher-level languages are translated into assembly language, which is a group of simple instructions. Sometimes performance- or timing-critical sections of code are written in assembly to guarantee specific timing or reduce computation time. This lab shows you how to create a RISC-V assembly language program that you can run on RVfpga using PlatformIO. We first give a brief overview of RISC-V assembly and then show how to create and run an assembly program on RVfpga. Then we provide exercises for you to practice writing your own RISC-V assembly programs.

2. RISC-V Assembly Language Overview

RISC-V assembly language includes simple instructions that are used to implement higher-level code. For example, some common RISC-V instructions include the `add`, `sub`, and `mul` instructions that add, subtract or multiply two operands.

The basic types of RISC-V instructions are: computational (arithmetic, logical, and shift) instructions, memory operations, and branches/jumps. The most common RISC-V instructions are given in Table 1. Instructions use operands that are located in registers or memory or that are encoded as a constant (i.e., *immediate*). RISC-V includes 32 32-bit registers. Table 2 lists the names of the 32 RISC-V registers. They can be specified by either their name (for example, `zero`, `s0`, `t5`, etc.) or their register number (i.e., `x0`, `x8`, `x30`). Programmers typically use register names which retains some information about the typical purpose of the register. For example, the saved registers, `s0`-`s11`, are typically used for program variables, while the temporary registers, `t0`-`t6` are used for temporary calculations. The `zero` register (`x0`) always contains the value 0, as this is a value commonly needed in programs. The other registers have specific uses as well, as shown in Table 2, but in this lab, you need only use the `zero` register and the temporary and saved registers.

Table 1. Common RISC-V assembly instructions

	RISC-V Assembly	Description	Operation
Computational	`add s0, s1, s2`	Add	`s0 = s1 + s2`
	`sub s0, s1, s2`	Subtract	`s0 = s1 - s2`
	`addi t3, t1, -10`	Add immediate	`t3 = t1 - 10`
	`mul t0, t2, t3`	32-bit multiply	`t0 = t2 * t3`
	`div s9, t5, t6`	Division	`t9 = t5 / t6`
	`rem s4, s1, s2`	Remainder	`s4 = s1 % s2`
	`and t0, t1, t2`	Bit-wise AND	`t0 = t1 & t2`
	`or t0, t1, t5`	Bit-wise OR	`t0 = t1 | t5`
	`xor s3, s4, s5`	Bit-wise XOR	`s3 = s4 ^ s5`
	`andi t1, t2, 0xFFB`	Bit-wise AND immediate	`t1 = t2 & 0xFFFFFFFB`
	`ori t0, t1, 0x2C`	Bit-wise OR immediate	`t0 = t1 | 0x2C`
	`xori s3, s4, 0xABC`	Bit-wise XOR immediate	`s3 = s4 ^ 0xFFFFFABC`
	`sll t0, t1, t2`	Shift left logical	`t0 = t1 << t2`
	`srl t0, t1, t5`	Shift right logical	`t0 = t1 >> t5`
	`sra s3, s4, s5`	Shift right arithmetic	`s3 = s4 >>> s5`
	`slli t1, t2, 30`	Shift left logical immediate	`t1 = t2 << 30`
	`srli t0, t1, 5`	Shift right logical immediate	`t0 = t1 >> 5`
	`srai s3, s4, 31`	Shift right arithmetic immediate	`s3 = s4 >>> 31`

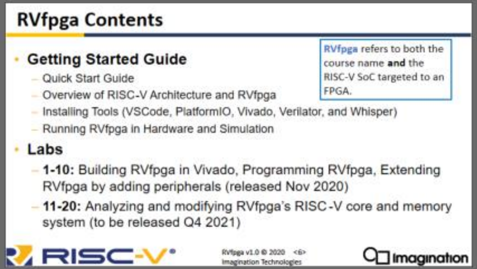
강사용 교재
10개 예제 해설

교육 자료 파일 12

RVfpga Course Contents

RVfpga v1.0 © 2020 <5> Imagination Technologies

5



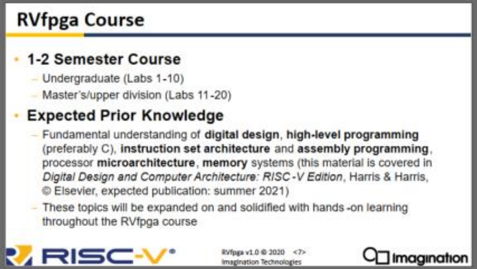
RVfpga Contents

RVfpga refers to both the course name **and** the RISC-V SoC targeted to an FPGA.

- **Getting Started Guide**
  - Quick Start Guide
  - Overview of RISC-V Architecture and RVfpga
  - Installing Tools (VSCode, PlatformIO, Vivado, Verilator, and Whisper)
  - Running RVfpga in Hardware and Simulation
- **Labs**
  - **1-10:** Building RVfpga in Vivado, Programming RVfpga, Extending RVfpga by adding peripherals (released Nov 2020)
  - **11-20:** Analyzing and modifying RVfpga's RISC-V core and memory system (to be released Q4 2021)

RVfpga v1.0 © 2020 <6> Imagination Technologies

6

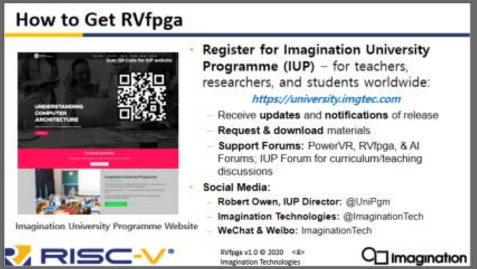


RVfpga Course

- **1-2 Semester Course**
  - Undergraduate (Labs 1-10)
  - Master's/upper division (Labs 11-20)
- **Expected Prior Knowledge**
  - Fundamental understanding of **digital design**, **high-level programming** (preferably C), **instruction set architecture** and **assembly programming**, processor **microarchitecture**, **memory** systems (this material is covered in *Digital Design and Computer Architecture: RISC-V Edition*, Harris & Harris, © Elsevier, expected publication: summer 2021)
  - These topics will be expanded on and solidified with hands-on learning throughout the RVfpga course

RVfpga v1.0 © 2020 <7> Imagination Technologies

7



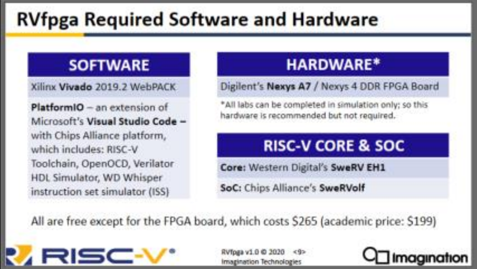
How to Get RVfpga

Imagination University Programme Website

- **Register for Imagination University Programme (IUP)** – for teachers, researchers, and students worldwide: *https://university.imgtec.com*
  - Receive **updates** and **notifications** of release
  - **Request & download** materials
  - **Support Forums:** PowerVR, RVfpga, & AI Forums; IUP Forum for curriculum/teaching discussions
- **Social Media:**
  - **Robert Owen, IUP Director:** @UniPgm
  - **Imagination Technologies:** @ImaginationTech
  - **WeChat & Weibo:** ImaginationTech

RVfpga v1.0 © 2020 <8> Imagination Technologies

8



RVfpga Required Software and Hardware

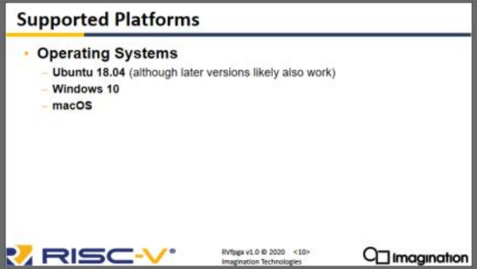
SOFTWARE	HARDWARE*
Xilinx **Vivado** 2019.2 WebPACK	Digilent's **Nexys A7** / Nexys 4 DDR FPGA Board
PlatformIO – an extension of Microsoft's **Visual Studio Code** – with Chips Alliance platform, which includes: RISC-V Toolchain, OpenOCD, Verilator HDL Simulator, WD Whisper instruction set simulator (ISS)	*All labs can be completed in simulation only; so this hardware is recommended but not required.

RISC-V CORE & SOC
Core: Western Digital's **SweRV EH1**
SoC: Chips Alliance's **SweRVolf**

All are free except for the FPGA board, which costs $265 (academic price: $199)

RVfpga v1.0 © 2020 <9> Imagination Technologies

9

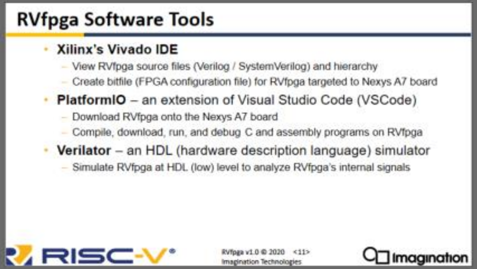


Supported Platforms

- **Operating Systems**
  - **Ubuntu 18.04** (although later versions likely also work)
  - **Windows 10**
  - **macOS**

RVfpga v1.0 © 2020 <10> Imagination Technologies

10

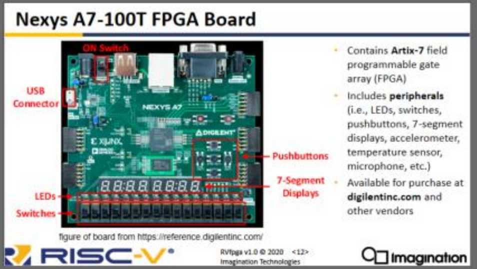


RVfpga Software Tools

- **Xilinx's Vivado IDE**
  - View RVfpga source files (Verilog / SystemVerilog) and hierarchy
  - Create bitfile (FPGA configuration file) for RVfpga targeted to Nexys A7 board
- **PlatformIO** – an extension of Visual Studio Code (VSCode)
  - Download RVfpga onto the Nexys A7 board
  - Compile, download, run, and debug C and assembly programs on RVfpga
- **Verilator** – an HDL (hardware description language) simulator
  - Simulate RVfpga at HDL (low) level to analyze RVfpga's internal signals

RVfpga v1.0 © 2020 <11> Imagination Technologies

11



Nexys A7-100T FPGA Board

ON Switch
USB Connector
Pushbuttons
7-Segment Displays
LEDs
Switches

figure of board from https://reference.digilentinc.com/

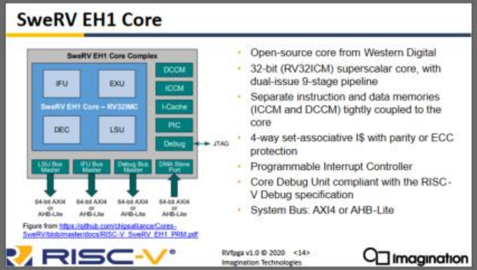
- Contains **Artix-7** field programmable gate array (FPGA)
- Includes **peripherals** (i.e., LEDs, switches, pushbuttons, 7-segment displays, accelerometer, temperature sensor, microphone, etc.)
- Available for purchase at **digilentinc.com** and other vendors

RVfpga v1.0 © 2020 <12> Imagination Technologies

12

RISC-V Cores and SoCs

RVfpga v1.0 © 2020 <13> Imagination Technologies



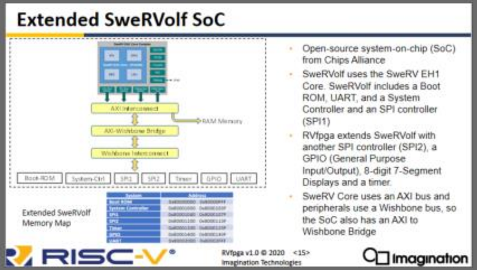
SweRV EH1 Core

SweRV EH1 Core Complex
IFU
EXU
SweRV EH1 Core – RV32IMC
DEC
LSU
DCCM
ICCM
I-Cache
PIC
Debug
JTAG
LSU Bus Master
IFU Bus Master
Debug Bus Master
DMA Slave Port
64-bit AXI4 or AHB-Lite

Figure from https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf

- Open-source core from Western Digital
- 32-bit (RV32ICM) superscalar core, with dual-issue 9-stage pipeline
- Separate instruction and data memories (ICCM and DCCM) tightly coupled to the core
- 4-way set-associative I$ with parity or ECC protection
- Programmable Interrupt Controller
- Core Debug Unit compliant with the RISC-V Debug specification
- System Bus: AXI4 or AHB-Lite

RVfpga v1.0 © 2020 <14> Imagination Technologies



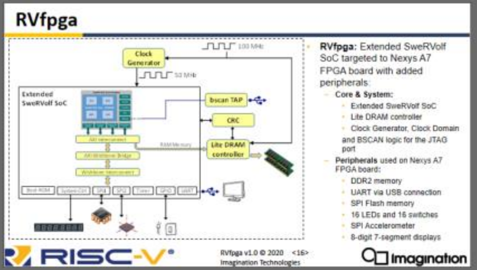
Extended SweRVolf SoC

AXI Interconnect
RAM Memory
AXI-Wishbone Bridge
Wishbone Interconnect
Boot ROM
System-Ctrl
SPI1
SPI2
Timer
GPIO
UART

Extended SweRVolf Memory Map

- Open-source system-on-chip (SoC) from Chips Alliance
- SweRVolf uses the SweRV EH1 Core. SweRVolf includes a Boot ROM, UART, and a System Controller and an SPI controller (SPI1)
- RVfpga extends SweRVolf with another SPI controller (SPI2), a GPIO (General Purpose Input/Output), 8-digit 7-Segment Displays and a timer.
- SweRV Core uses an AXI bus and peripherals use a Wishbone bus, so the SoC also has an AXI to Wishbone Bridge

RVfpga v1.0 © 2020 <15> Imagination Technologies



RVfpga

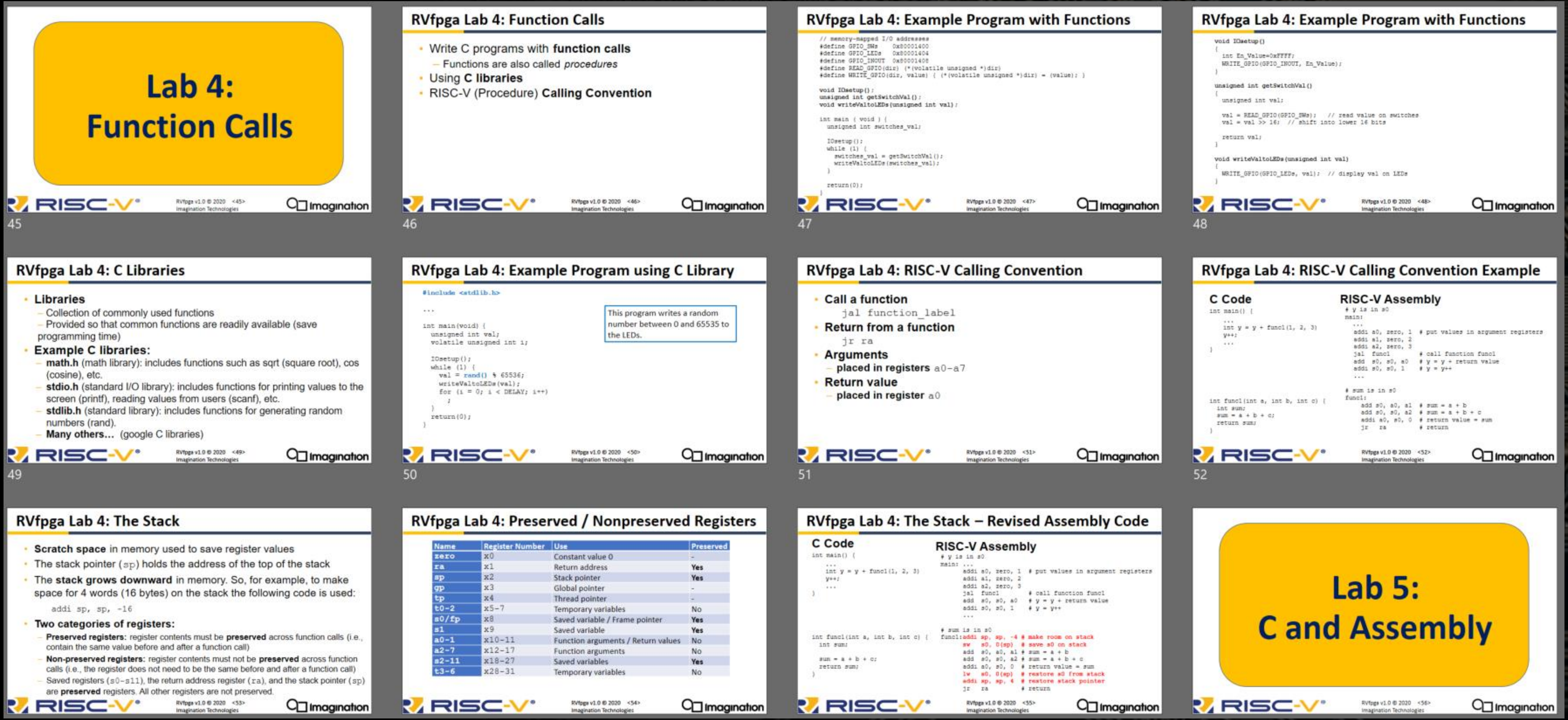
100 MHz
Clock Generator
50 MHz
Extended SweRVolf SoC
bscan TAP
CRC
Lite DRAM controller
RAM Memory

- **RVfpga:** Extended SweRVolf SoC targeted to Nexys A7 FPGA board with added peripherals
  - **Core & System:**
    - Extended SweRVolf SoC
    - Lite DRAM controller
    - Clock Generator, Clock Domain and BSCAN logic for the JTAG port
  - **Peripherals** used on Nexys A7 FPGA board:
    - DDR2 memory
    - UART via USB connection
    - SPI Flash memory
    - 16 LEDs and 16 switches
    - SPI Accelerometer
    - 8-digit 7-segment displays

RVfpga v1.0 © 2020 <16> Imagination Technologies

슬라이드 파일

교육 자료 파일 13



45

Lab 4: Function Calls

RVfpga v1.0 © 2020 <45> Imagination Technologies

46

RVfpga Lab 4: Function Calls

- Write C programs with **function calls**
  - Functions are also called *procedures*
- Using **C libraries**
- RISC-V (Procedure) **Calling Convention**

RVfpga v1.0 © 2020 <46> Imagination Technologies

47

RVfpga Lab 4: Example Program with Functions

```
// memory-mapped I/O addresses
#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408
#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

void IOsetup();
unsigned int getSwitchVal();
void writeValtoLEDs(unsigned int val);

int main ( void ) {
  unsigned int switches_val;

  IOsetup();
  while (1) {
    switches_val = getSwitchVal();
    writeValtoLEDs(switches_val);
  }

  return(0);
}
```

RVfpga v1.0 © 2020 <47> Imagination Technologies

48

RVfpga Lab 4: Example Program with Functions

```
void IOsetup()
{
  int En_Value=0xFFFF;
  WRITE_GPIO(GPIO_INOUT, En_Value);
}

unsigned int getSwitchVal()
{
  unsigned int val;

  val = READ_GPIO(GPIO_SWs);   // read value on switches
  val = val >> 16;  // shift into lower 16 bits

  return val;
}

void writeValtoLEDs(unsigned int val)
{
  WRITE_GPIO(GPIO_LEDs, val);  // display val on LEDs
}
```

RVfpga v1.0 © 2020 <48> Imagination Technologies

49

RVfpga Lab 4: C Libraries

- **Libraries**
  - Collection of commonly used functions
  - Provided so that common functions are readily available (save programming time)
- **Example C libraries:**
  - **math.h** (math library): includes functions such as sqrt (square root), cos (cosine), etc.
  - **stdio.h** (standard I/O library): includes functions for printing values to the screen (printf), reading values from users (scanf), etc.
  - **stdlib.h** (standard library): includes functions for generating random numbers (rand).
  - **Many others…** (google C libraries)

RVfpga v1.0 © 2020 <49> Imagination Technologies

50

RVfpga Lab 4: Example Program using C Library

```
#include <stdlib.h>

...

int main(void) {
  unsigned int val;
  volatile unsigned int i;

  IOsetup();
  while (1) {
    val = rand() % 65536;
    writeValtoLEDs(val);
    for (i = 0; i < DELAY; i++)
      ;
  }
  return(0);
}
```

This program writes a random number between 0 and 65535 to the LEDs.

RVfpga v1.0 © 2020 <50> Imagination Technologies

51

RVfpga Lab 4: RISC-V Calling Convention

- **Call a function**
  `jal function_label`
- **Return from a function**
  `jr ra`
- **Arguments**
  - **placed in registers** `a0-a7`
- **Return value**
  - **placed in register** `a0`

RVfpga v1.0 © 2020 <51> Imagination Technologies

52

RVfpga Lab 4: RISC-V Calling Convention Example

C Code

```
int main() {
    ...
    int y = y + func1(1, 2, 3)
    y++;
    ...
}

int func1(int a, int b, int c) {
  int sum;
  sum = a + b + c;
  return sum;
}
```

RISC-V Assembly

```
# y is in s0
main:
  ...
  addi a0, zero, 1  # put values in argument registers
  addi a1, zero, 2
  addi a2, zero, 3
  jal  func1        # call function func1
  add  s0, s0, a0   # y = y + return value
  addi s0, s0, 1    # y = y++
  ...

# sum is in s0
func1:
    add s0, a0, a1  # sum = a + b
    add s0, s0, a2  # sum = a + b + c
    addi a0, s0, 0  # return value = sum
    jr   ra         # return
```

RVfpga v1.0 © 2020 <52> Imagination Technologies

RVfpga Lab 4: The Stack

- **Scratch space** in memory used to save register values
- The stack pointer (`sp`) holds the address of the top of the stack
- The **stack grows downward** in memory. So, for example, to make space for 4 words (16 bytes) on the stack the following code is used:
  `addi sp, sp, -16`
- **Two categories of registers:**
  - **Preserved registers:** register contents must be **preserved** across function calls (i.e., contain the same value before and after a function call)
  - **Non-preserved registers:** register contents must not be **preserved** across function calls (i.e., the register does not need to be the same before and after a function call)
  - Saved registers (`s0-s11`), the return address register (`ra`), and the stack pointer (`sp`) are **preserved** registers. All other registers are not preserved.

RVfpga v1.0 © 2020 <53> Imagination Technologies

RVfpga Lab 4: Preserved / Nonpreserved Registers

Name	Register Number	Use	Preserved
zero	x0	Constant value 0	-
ra	x1	Return address	Yes
sp	x2	Stack pointer	Yes
gp	x3	Global pointer	-
tp	x4	Thread pointer	-
t0-2	x5-7	Temporary variables	No
s0/fp	x8	Saved variable / Frame pointer	Yes
s1	x9	Saved variable	Yes
a0-1	x10-11	Function arguments / Return values	No
a2-7	x12-17	Function arguments	No
s2-11	x18-27	Saved variables	Yes
t3-6	x28-31	Temporary variables	No

RVfpga v1.0 © 2020 <54> Imagination Technologies

RVfpga Lab 4: The Stack – Revised Assembly Code

C Code

```
int main() {
    ...
    int y = y + func1(1, 2, 3)
    y++;
    ...
}

int func1(int a, int b, int c) {
  int sum;

  sum = a + b + c;
  return sum;
}
```

RISC-V Assembly

```
# y is in s0
main: ...
      addi a0, zero, 1  # put values in argument registers
      addi a1, zero, 2
      addi a2, zero, 3
      jal  func1        # call function func1
      add  s0, s0, a0   # y = y + return value
      addi s0, s0, 1    # y = y++
      ...

# sum is in s0
func1:addi sp, sp, -4 # make room on stack
      sw   s0, 0(sp)  # save s0 on stack
      add  s0, a0, a1 # sum = a + b
      add  s0, s0, a2 # sum = a + b + c
      addi a0, s0, 0  # return value = sum
      lw   s0, 0(sp)  # restore s0 from stack
      addi sp, sp, 4  # restore stack pointer
      jr   ra         # return
```

RVfpga v1.0 © 2020 <55> Imagination Technologies

Lab 5: C and Assembly

RVfpga v1.0 © 2020 <56> Imagination Technologies

슬라이드 파일, 113 pages

IUP 등록 및 교육자료 다운로드

- https://university.imgtec.com
- 등록 하시면 승인 이메일을 보내 드립니다.
- 근무시간 기준 72시간내에 승인 발송

- https://university.imgtec.com/teaching-download
- 승인 이메일을 확인 후, 다운로드 받으시면 됩니다.

IUP 교육자료에 관심을 가져 주셔서 감사합니다.

https://university.imgtec.com