

RVfpga

컴퓨터 아키텍처 이해 과정

Acknowledgements

저자

Prof. Sarah Harris
Prof. Daniel Chaver
Zubair Kakakhel
M. Hamza Liaqat

고문

Prof. David Patterson

기여자

Robert Owen
Olof Kindgren
Prof. Luis Piñuel
Ivan Kravets
Valerii Koval
Ted Marena
Prof. Roy Kravitz

ASSOCIATES

Prof. José Ignacio Gómez	Prof. Francisco Tirado	Gage Elerding
Prof. Christian Tenllado	Prof. Román Hermida	Prof. Brian Cruickshank
Prof. Daniel León	Prof. Ataur Patwary	Deepen Parmar
Prof. Katzalin Olcoz	Cathal McCabe	Thong Doan
Prof. Alberto del Barrio	Dan Hugo	Oliver Rew
Prof. Fernando Castro	Braden Harwood	Niko Nikolay
Prof. Manuel Prieto	Prof. David Burnett	Guanyang He

Sponsors and Supporters

Western Digital

 **Imagination**

 **CHIPS
ALLIANCE**

 **RISC-V**

 **DIGILENT**
A National Instruments Company

 **XILINX**
| UNIVERSITY PROGRAM

 **Digi-Key**
ELECTRONICS

 **Esperanto**
TECHNOLOGIES

 **codasip**


硬禾学堂

 **ANDES**
TECHNOLOGY

 **PLATFORMIO.ORG**

소개

- RISC-V FPGA (**RVfpga**)는 아래 과정에 대한 tool 및 LAB을 제공하는 교육 패키지입니다.
 - 상용 RISC-V SoC 를 **FPGA**에 적용
 - RISC-V SoC **프로그래밍**
 - RISC-V SoC에 **더 많은 기능 추가**
 - RISC-V 코어 및 메모리 계층 **분석 및 수정**
- 이 패키지는 **Imagination Technologies**와 학교 및 업계 파트너가 개발 중입니다.
- RVfpga 시스템은 Western Digital의 RISC-V SweRV EH1 코어를 기반으로 하는 Chips Alliance의 SweRVolf SoC를 기반으로 구축되었습니다.

RVfpga 개요

- **RVfpga 패키지는** 다음을 제공합니다:
 - 포괄적이고 자유롭게 배포되는 완전한 RISC-V 과정
 - RISC-V 프로세서 및 RISC-V 에코 시스템에 대해 **쉽게 공부하고 접근할 수 있는 방법**
 - **저렴한 FPGA**를 대상으로 하는 RISC-V 시스템으로 많은 대학과 회사에서 쉽게 적용할 수 있습니다.
- RVfpga 과정을 마친 사용자는 RISC-V 프로세서, SoC 및 에코 시스템을 사용하는 방법을 이해하고, 응용할 수 있습니다.

RVfpga 코스 내용



RVfpga 내용

- 시작 가이드

- 빠른 시작 가이드
- RISC-V 아키텍처 및 RVfpga 개요
- Tool 설치 (VSCode, PlatformIO, Vivado, Verilator 및 Whisper)
- 하드웨어 및 시뮬레이션에서 RVfpga 시스템 실행

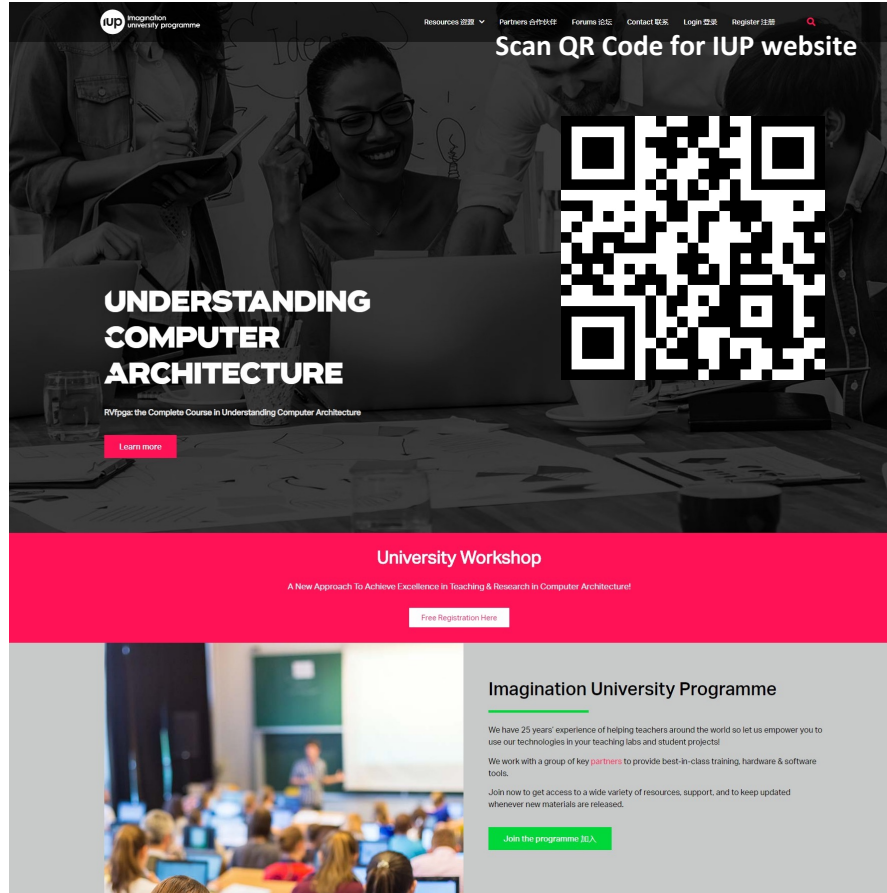
- Labs

- **1-10:** Vivado에서 RVfpga 빌드, RVfpga 프로그래밍, 주변 장치를 추가하여 RVfpga 확장 (2020 년 11 월 출시)
- **11-20:** RVfpga의 RISC-V 코어 및 메모리 시스템 분석 및 수정 (2021 년 4 분기 출시 예정)

RVfpga 코스

- 2-4 학기 과정
 - 학부 (LABS 1-10)
 - 석사 / 상위(Labs 11-20)
- 요구되는 사전 지식
 - 디지털 설계, 고급 프로그래밍 (C 권장), 명령어 세트 아키텍처 및 어셈블리 프로그래밍, 프로세서 마이크로 아키텍처, 메모리 시스템에 대한 기본 이해 (이 자료는 *Digital Design and Computer Architecture: RISC-V Edition*, Harris & Harris, © Elsevier 에서 다룹니다. 2021 년 여름 출판 예정)
 - 이 주제는 RVfpga 과정 전체에서 LAB 학습을 통해 확장되고 강화됩니다.

RVfpga를 구하는 방법



Imagination University Programme Website

- IUP (Imagination University Program) 등록 – 전 세계의 교사, 연구원 및 학생용:

<https://university.imgtec.com>

- 업데이트 및 릴리스 알림 받기
 - 자료 요청 및 다운로드
 - 지원 포럼: PowerVR, RVfpga 및 AI 포럼; 커리큘럼/교수 토론을 위한 IUP 포럼
-
- 소셜 미디어:
 - Robert Owen, IUP Director: @UniPgm
 - Imagination Technologies: @ImaginationTech
 - WeChat & Weibo: ImaginationTech

RVfpga 필수 소프트웨어 및 하드웨어

소프트웨어

Xilinx Vivado 2019.2 WebPACK

PlatformIO – Microsoft **Visual Studio Code**의 확장 – Chips Alliance 플랫폼 포함: RISC-V Toolchain, OpenOCD, Verilator HDL Simulator, WD Whisper ISS (명령어 집합 시뮬레이터)

하드웨어*

Digilent Nexys A7 / Nexys 4 DDR FPGA Board

* 모든 LAB은 시뮬레이션에서만 완료할 수 있습니다. 따라서 이 하드웨어는 권장되지만 필수는 아닙니다.

RISC-V 코어 및 SOC

Core: Western Digital의 SweRV EH1

SoC: Chips Alliance의 SweRVolf

\$ 265 (교육 할인: \$ 199) 인 FPGA 보드를 제외하고 모두 무료입니다.

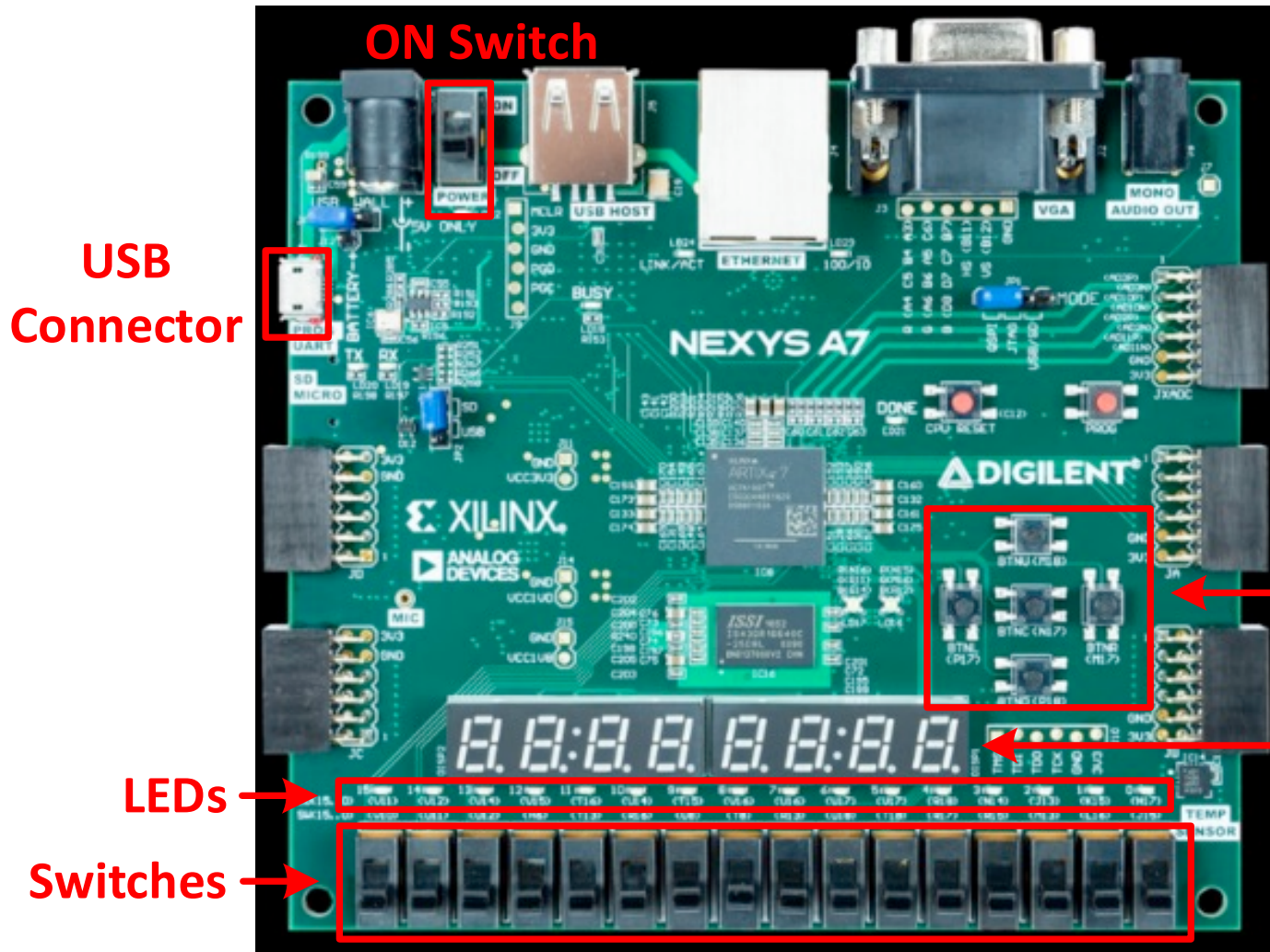
지원 플랫폼

- 운영체제
 - Ubuntu 18.04 (향후 버전도 작동할 가능성이 있음)
 - Windows 10
 - macOS

RVfpga 소프트웨어 Tool

- **Xilinx의 Vivado IDE**
 - RVfpga 소스 파일 (Verilog/SystemVerilog) 및 계층 보기
 - Nexys A7 보드를 대상으로 하는 RVfpga 용 비트 파일 (FPGA 구성 파일) 생성
- **PlatformIO – Visual Studio Code (VSCode의 확장)**
 - Nexys A7 보드에 RVfpga 시스템 다운로드
 - Rvfpga 시스템에서 C 및 어셈블리 프로그램 컴파일, 다운로드, 실행 및 디버그
 -
- **Verilator – HDL (하드웨어 설명 언어) 시뮬레이터**
 - HDL (low) 수준에서 RVfpga 시스템을 시뮬레이션하여 내부 신호 분석

Nexys A7-100T FPGA 보드



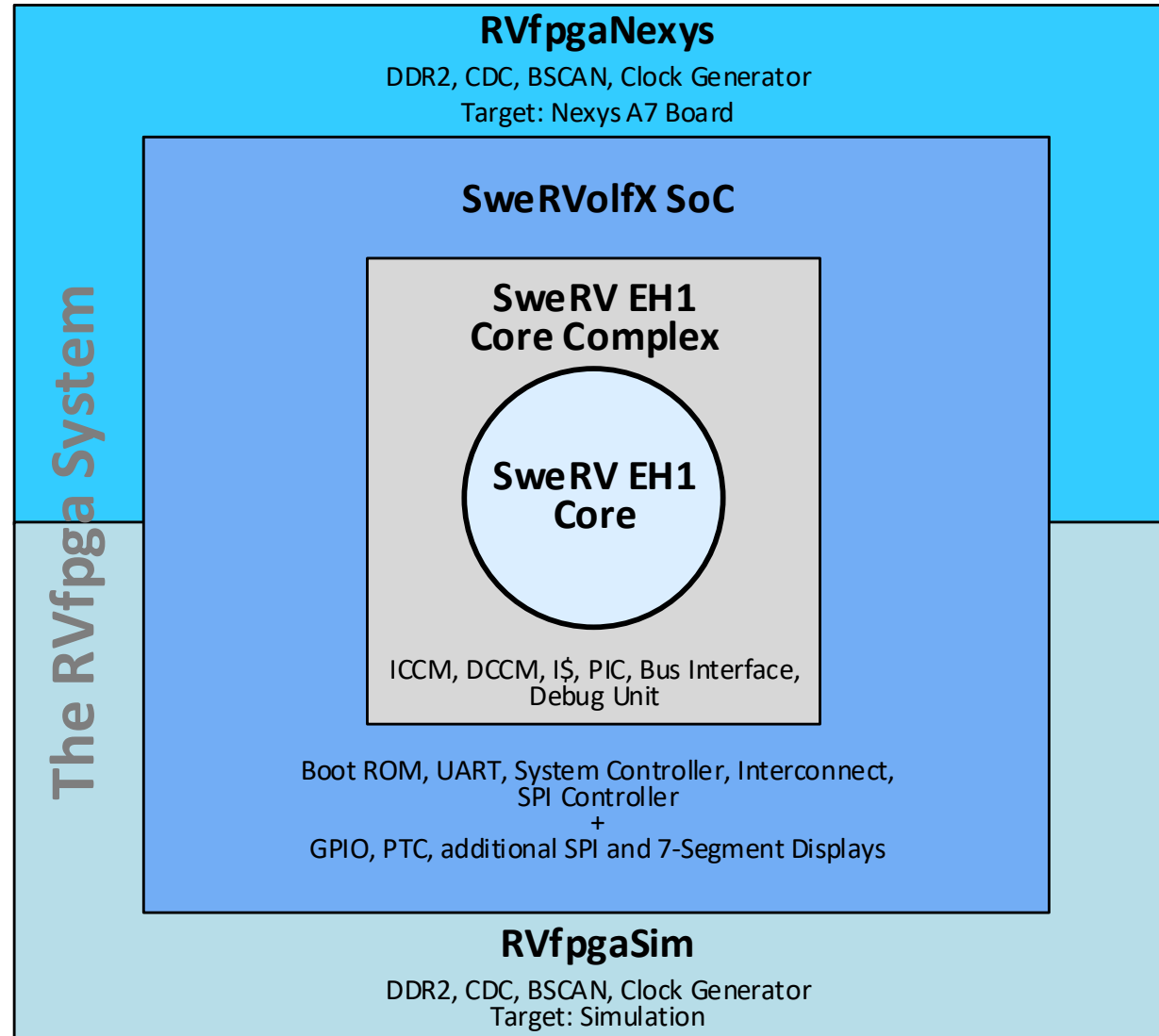
- Artix-7 필드 프로그래밍 가능 게이트 어레이 (FPGA)
- 주변 장치 (예: LED, 스위치, 푸시 버튼, 7 세그먼트 디스플레이, 가속도계, 온도 센서, 마이크 등) 포함
- digilentinc.com 및 기타 공급 업체에서 구매 가능

<https://reference.digilentinc.com/> 의 보드 그림

RISC-V 코어 및 SoCs



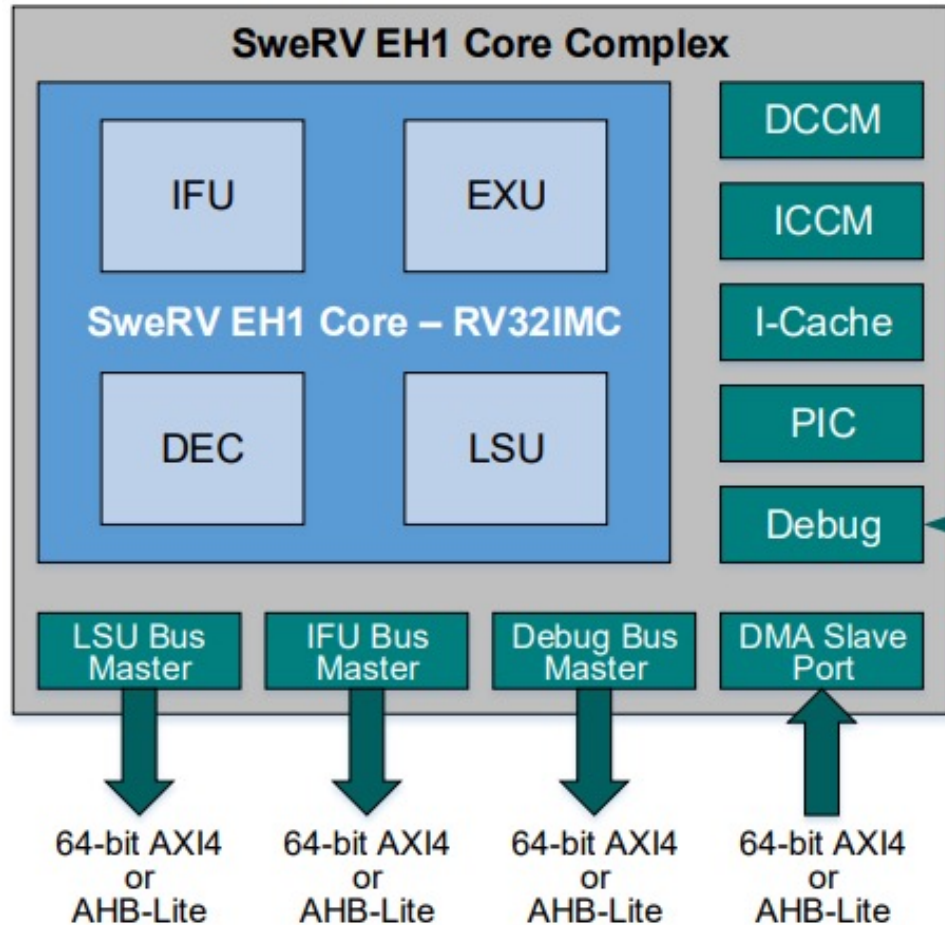
RVfpga Hierarchy



RVfpga Hierarchy

Name	Description
SweRV EH1 Core	Open-source commercial RISC-V core developed by Western Digital (https://github.com/chipsalliance/Cores-SweRV).
SweRV EH1 Core Complex	SweRV EH1 core with added memory (ICCM, DCCM, and instruction cache), programmable interrupt controller (PIC), bus interfaces, and debug unit (https://github.com/chipsalliance/Cores-SweRV).
SweRVolfX (Extended SweRVolf)	The System on Chip that we use in the RVfpga course. It is an extension of SweRVolf. <u>SweRVolf</u> (https://github.com/chipsalliance/Cores-SweRVolf): An open-source SoC built around the SweRV EH1 Core Complex. It adds a boot ROM, UART interface, system controller, interconnect (AXI Interconnect, Wishbone Interconnect, and AXI-to-Wishbone bridge), and an SPI controller. <u>SweRVolfX</u> : It adds 4 new peripherals to SweRVolf: a GPIO, a PTC, an additional SPI and a controller for the 8 digit 7-Segment Displays.
RVfpgaNexys	The SweRVolfX SoC targeted to the Nexys A7 board and its peripherals. It adds a DDR2 interface, CDC (clock domain crossing) unit, BSCAN logic (for the JTAG interface), and clock generator. RVfpgaNexys is the same as SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), except that the latter is based on SweRVolf.
RVfpgaSim	The SweRVolfX SoC with a testbench wrapper and AXI memory intended for simulation. RVfpgaSim is the same as SweRVolf sim, (https://github.com/chipsalliance/Cores-SweRVolf), except that the latter is based on SweRVolf.

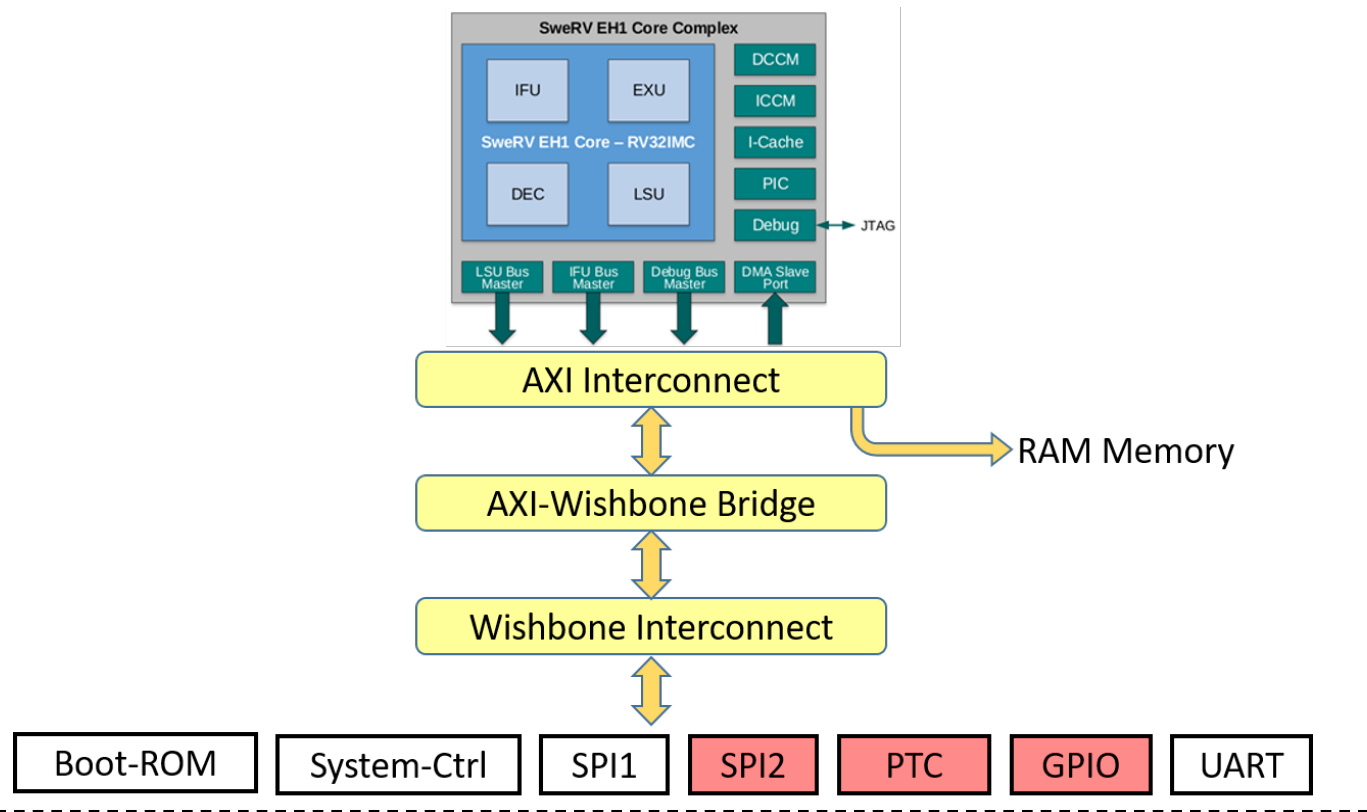
SweRV EH1 Core and SweRV EH1 Core Complex



- Western Digital의 오픈 소스 코어
- 32 비트 (RV32ICM) 슈퍼 스칼라 코어, dual-issue 9-stage pipeline
- 코어에 밀접하게 결합된 별도의 명령 및 데이터 메모리 (ICCM 및 DCCM)
- 패리티 또는 ECC 보호 기능이 있는 4 방향 세트 연관 I\$
- 프로그래밍 가능한 인터럽트 컨트롤러
- RISC-V 디버그 사양을 준수하는 코어 디버그 장치
- 시스템 버스: AXI4 또는 AHB-Lite

그림 출처 [https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V SweRV EH1 PRM.pdf](https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V%20SweRV%20EH1%20PRM.pdf)

SweRVolfX SoC

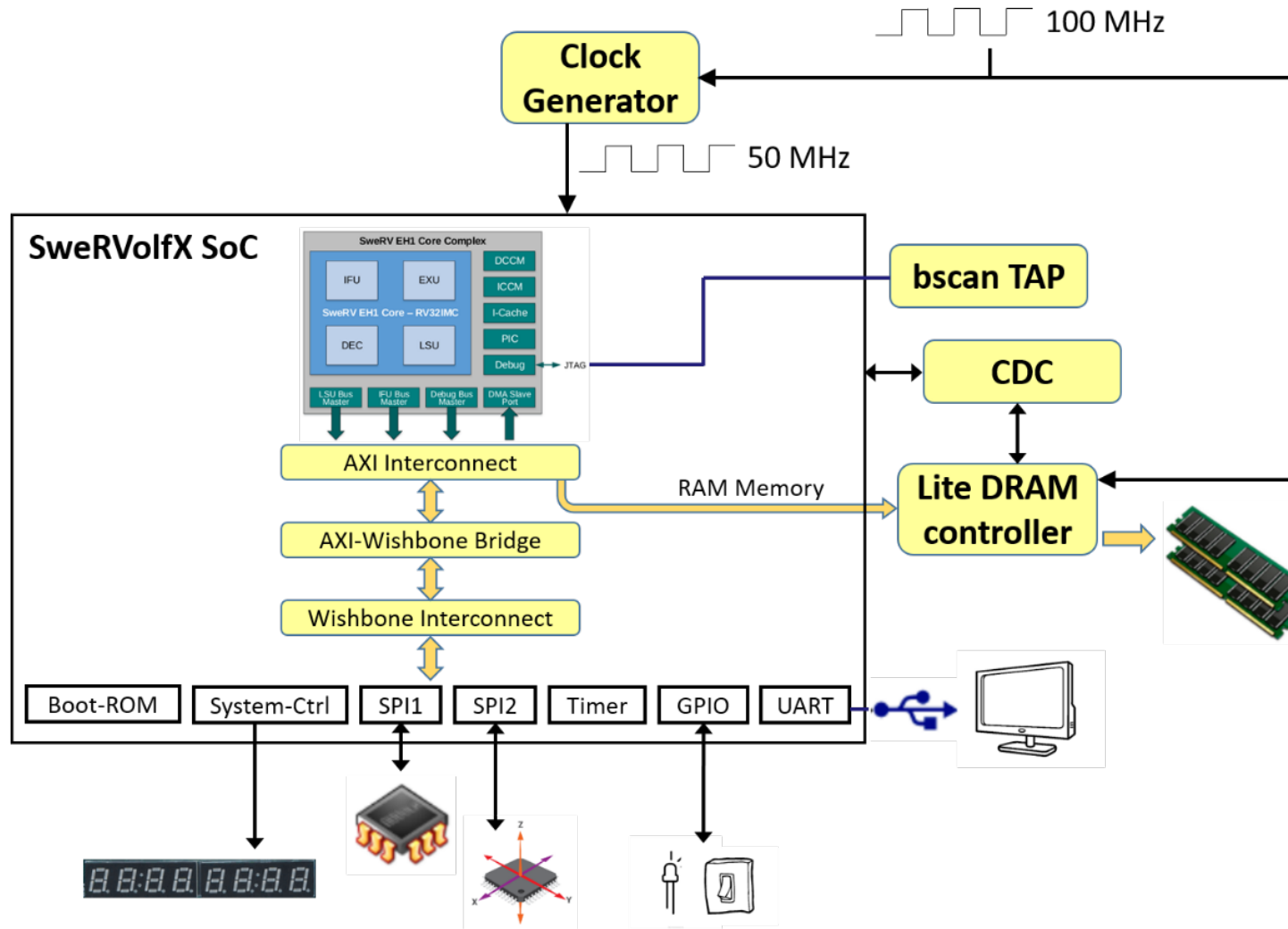


SweRVolfX 메모리 맵

System	Address
Boot ROM	0x80000000 - 0x80000FFF
System Controller	0x80001000 - 0x8000103F
SPI1	0x80001040 - 0x8000107F
SPI2	0x80001100 - 0x8000113F
Timer	0x80001200 - 0x8000123F
GPIO	0x80001400 - 0x8000143F
UART	0x80002000 - 0x80002FFF

- Chips Alliance의 오픈 소스 SoC (system-on-chip)
- SweRVolf는 SweRV EH1 Core를 사용합니다. SweRVolf에는 부팅 ROM, UART, 시스템 컨트롤러 및 SPI 컨트롤러 (SPI1)가 포함됩니다.
- SweRVolfX는 다른 SPI 컨트롤러 (SPI2), GPIO (범용 입/출력), 7세그먼트 디스플레이 및 PTC (빨간색으로 표시)로 SweRVolf를 확장합니다.
- SweRV EH1 Core는 AXI 버스를 사용하고 주변 장치는 Wishbone 버스를 사용하므로 SoC에는 AXI-Wishbone 브릿지도 있습니다.

RvfpgaNexys



- **RVfpgaNexys:** 주변 장치가 추가된 Nexys A7 FPGA 보드를 대상으로하는 SweRVolfX SoC:
 - **코어 및 시스템 :**
 - SweRVolf SoC
 - Lite DRAM 컨트롤러
 - 클럭 생성기, 클럭 도메인 및 JTAG 포트에 대한 BSCAN 로직
 - Nexys A7 FPGA 보드에 사용되는 **주변 장치 :**
 - DDR2 메모리
 - USB 연결을 통한 UART
 - SPI 플래시 메모리
 - 16 개의 LED 및 16 개의 스위치
 - SPI 가속도계
 - 8 자리 7 세그먼트 디스플레이

- SweRVolfX SoC는 Verilog 래퍼를 포함하여 시뮬레이션을 활성화 할 수도 있습니다.
- **RVfpgaSim**은 HDL 시뮬레이터에서 사용하기 위해 테스트 벤치에 래핑된 SweRVolfX SoC입니다.

RVfpga 시스템 확장

- Rvfpga System 은 Labs 6-10에서 추가로 확장되었습니다:
 - 온보드 Nexys A7 푸시 버튼과 인터페이스하는 GPIO 컨트롤러
 - 7 세그먼트 디스플레이 컨트롤러 수정
 - 온보드 3 색 LED 사용을 위한 새로운 타이머 모듈
 - 새로운 외부 인터럽트 소스

RVfpga Labs 개요

RVfpga Labs

Labs 1-10 (릴리스 날짜: 2020년 11월)

- Vivado 프로젝트 및 프로그래밍
- I/O 시스템

Labs 11-20 (2021년 4분기 출시 예정)

- RISC-V 코어
- RISC-V 메모리 시스템

모든 LABS에는 이해도를 높이기 위해 Rvfpga 시스템을 사용하고 수정하는 실습이 포함되어 있습니다.

RVfpga Labs 1-10

RVfpga 시스템 소스 코드 (Verilog / SV)를 보고 이를 FPGA (Lab 1)에 대상으로 지정하는 방법, RVfpga 용 C 및 어셈블리 프로그램 (Labs 2-5)을 작성하고 RVfpgaNexys/RVfpgaSim 를 수정하여 주변 장치를 추가하는 방법을 (Labs 6-10) 보여줍니다.

- Lab 0: RVfpga Labs 개요
- Lab 1: Vivado 프로젝트 생성
- Lab 2: C 프로그래밍
- Lab 3: RISC-V 어셈블리 언어
- Lab 4: 함수 호출
- Lab 5: 이미지 처리: C 및 어셈블리
- Lab 6: I/O 소개
- Lab 7: 7 세그먼트 디스플레이
- Lab 8: 타이머
- Lab 9: 인터럽트 구동 I/O
- Lab 10: 직렬 버스

프로그래밍

I/O 시스템

RVfpga Labs 1-5: Vivado 프로젝트 및 프로그래밍

- **Lab 1: Vivado 프로젝트 생성:** RVfpgaNexys를 FPGA 보드로 타겟팅하고 Verilator에서 RVfpgaSim을 시뮬레이션하는 Vivado 프로젝트를 빌드합니다.
- **Lab 2: C 프로그래밍:** PlatformIO에서 C 프로그램을 작성하고 RVfpgaNexys/RVfpgaSim/Whisper 에서 실행/디버그 합니다. 또한 터미널 인쇄와 같은 작업을 지원하기 위해 Western Digital의 보드 지원 및 플랫폼 지원 패키지 (BSP 및 PSP)를 소개합니다.
- **Lab 3: RISC-V 어셈블리 언어:** PlatformIO에서 RISC-V 어셈블리 프로그램을 작성하고 RVfpgaNexys/RVfpgaSim/Whisper 에서 실행 / 디버그
- **Lab 4: 함수 호출:** 함수 호출 소개, C 라이브러리 및 RISC-V 호출 규칙
- **Lab 5: 이미지 처리: C 및 어셈블리:** C 코드와 어셈블리 코드 포함

RVfpga Labs 6-10: I/O 및 주변기기

- LAB 6: I/O 소개: 메모리 매핑 I/O 및 Rvfpga 시스템의 오픈 소스 GPIO 모듈 소개
- LAB 7: 7 세그먼트 디스플레이: 7 세그먼트 디스플레이 디코더를 구축하고 이를 RVfpga 시스템에 통합
- LAB 8: 타이머: 타이머 및 타이머 컨트롤러 이해 및 사용
- LAB 9: 인터럽트 구동 I/O: RVfpga 시스템 인터럽트 지원 소개 및 인터럽트 구동 I/O 사용
- LAB 10: 직렬 버스: 직렬 인터페이스 (SPI, I2C, UART) 소개. SPI 인터페이스를 사용하는 온보드 가속도계 사용

RVfpga Labs 11-15: The RISC-V Core

- 2021년 4분기 출시 예정
- 핵심 구조 이해
- 파이프 라인 (산술/논리, 메모리, 점프 및 분기)을 통한 명령 흐름 이해
- 위험요소 파악 및 대처 방법 이해
- 새로운 명령어 구현 및 FPGA 보드에서 실행
- Branch 예측기 이해 및 수정
- 슈퍼 스칼라 처리 이해

RVfpga Labs 16-20: RISC-V 메모리 시스템

- 2021 년 4 분기 출시 예정
- 캐시 적중 및 실패를 포함한 메모리 계층의 작동 이해
- 캐시 수정: 다양한 캐시 크기, 구성 및 관리 정책 구현
- 캐시 컨트롤러 이해
- 메모리 이해: ICCM (명령 밀접 결합 메모리) 및 DCCM (데이터 밀접 결합 메모리)

RVfpga 타임라인

RVfpga Availability

November 2020	RVfpga Getting Started Guide RVfpga Labs 1-10
Q4 2021	RVfpga Labs 11-20
June 2021	Masters-level SoC Design Course
Languages	English & Chinese (Spanish & Japanese to follow)
Textbook	<i>Digital Design & Computer Architecture: RISC-V Edition 2021</i> by Sarah Harris and David Harris

• 대상

- 전기 공학, 컴퓨터 과학 또는 컴퓨터 공학 학부생
- RISC-V 아키텍처 학습에 관심이 있는 학계 및 산업 전문가

• IUP (Imagination University Program) 실적: MIPSfpga 프로그램 개발:

- 2015 년 4 월 출시
- 800개 대학 참여
- 수상자: Elektra Best Educational Support Award, Europe 2015

RVfpga

빠른 시작 가이드


빠른 시작 가이드 개요

- VSCode 및 PlatformIO 설치
- RVfpgaNexys에서 예제 프로그램 실행

Install PlatformIO & VSCode

- VSCode 설치
 - <https://code.visualstudio.com/Download>
 - Ubuntu 및 macOS의 경우 Python을 설치합니다 (Windows에는 이 단계가 필요하지 않음).
- VSCode 내에 PlatformIO 확장 설치
- Nexys A7 보드 드라이버 설치 (RVfpga 시작 안내서 참조)

RVfpgaNexys를 보드에 다운로드하고 프로그램 실행

- PlatformIO에서:
 - 스위치 값을 LED에 쓰는 예제 프로그램을 엽니다. 프로그램 위치:
`[RVfpgaPath]#RVfpga#examples#LedsSwitches_C-Lang`
 - PlatformIO 초기화 파일 (platformio.ini)에서 **RVfpgaNexys bitfile**의 디렉토리 위치 업데이트 - 즉, 다음 행을 platformio.ini에 추가합니다. `board_build.bitstream_file = [RVfpgaPath]/RVfpga/src/rvfpga.bit`
 - Nexys A7 보드에 **RVfpgaNexys 다운로드** (프로젝트 작업 → env: swervolf_nexys → 플랫폼 → 비트 스트림 업로드)
 - 실행 / 디버그 버튼을 눌러 RVfpgaNexys에서 프로그램 컴파일, 다운로드 및 

[RVfpgaPath]는 컴퓨터에서 RVfpga 폴더의 위치입니다. 이 폴더는 Imagination University Program의 RVfpga 패키지와 함께 제공되었습니다.

RVfpga Labs

설명



Lab 1: Vivado 프로젝트



RVfpga Lab 1: RVfpga Vivado 프로젝트

- **Vivado**는 RVfpga 시스템용 소스 (Verilog) 코드를 보고, 수정하고, 합성하기 위한 Xilinx Tool입니다.
- Rvfpga 시스템의 소스 코드는 다음 위치에 있습니다:
[RVfpgaPath]/RVfpga/src
- Rvfpga 시스템의 소스 코드가 포함된 Vivado 프로젝트를 만듭니다. Nexys A7 보드를 대상으로 하는 RVfpgaNexys를 합성하고 FPGA를 RVfpgaNexys로 구성하기 위한 정보가 포함된 **bitfile**(bitstream 파일이라고도 함)을 생성합니다.
- HDL 시뮬레이터인 Verilator를 사용하여 Rvfpga 시스템의 소스 코드를 시뮬레이션하고 내부 신호를 검사 할 수도 있습니다 (Verilator 사용 방법에 대한 지침은 Rvfpga 시작 안내서 참조).
- Vivado와 Verilator는 Rvfpga 시스템을 수정하고 시뮬레이션하기 위해 RVfpga Labs 6-10에서 광범위하게 사용될 것입니다.

Lab 2:

C 프로그래밍



RVfpga Lab 2: C 프로그래밍

- PlatformIO 프로젝트 만들기
- 프로젝트에 예제 C 프로그램 추가
- Nexys A7 보드에 RVfpgaNexys 다운로드
- RVfpgaNexys에 C 프로그램 다운로드 및 프로그램 실행/디버그
- LAB 종료시 일부 또는 모든 연습 완료
- Verilator (using **RVfpgaSim**) or **Whisper** 에 있는 프로그램을 시뮬레이션 할 수 있습니다.

RVfpga Lab 2: C 프로그램 예시

```
// 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); }
```

```
int main ( void )
```

```
{
```

```
    int En_Value=0xFFFF, switches_value;
```

```
    WRITE_GPIO(GPIO_INOUT, En_Value);
```

```
    while (1) {
```

```
        switches_value = READ_GPIO(GPIO_SWs);    // read value on switches
```

```
        switches_value = switches_value >> 16;    // shift into lower 16 bits
```

```
        WRITE_GPIO(GPIO_LEDs, switches_value);    // display switch value on LEDs
```

```
    }
```

```
    return(0);
```

```
}
```

이 프로그램은 스위치의 값을 LED에 기록합니다.

RVfpga Lab 2: 메모리 매핑 된 I/O 주소

Device	Memory-Mapped I/O Address
스위치 (Nexys A7 보드의 16 개)	0x80001400 (upper 16 bits)
LEDs (Nexys A7 보드의 16 개)	0x80001404 (lower 16 bits)
GPIO의 입력/출력 (1 = 출력, 0 = 입력)	0x80001408

RVfpga Lab 2: Western Digital의 BSP 및 PSP

- Western Digital은 다음을 제공합니다:
 - **PSP:** 프로세서 지원 패키지
 - **BSP:** 보드 지원 패키지
- 이들은 주어진 프로세서 (SweRV EH1 코어) 및 보드 (Nexys A7 FPGA 보드)에 대한 공통 기능을 제공합니다.
예제: `printfNexys` (C의 `printf` 함수처럼)

RVfpga Lab 2: Using UART to Print to Terminal

```
#if defined(D_NEXYS_A7)
    #include <bsp_printf.h>
    #include <bsp_mem_map.h>
    #include <bsp_version.h>
#else
    PRE_COMPILED_MSG("no platform was defined")
#endif
#include <psp_api.h>
#define DELAY 10000000

int main(void) {
    int i, j = 0;

    // Initialize UART
    uartInit();
    while (1) {
        printfNexys("Hello RVfpga users! Iteration: %d\n", j);
        for (i=0; i < DELAY; i++) ; // delay between printf's
        j++;
    }
}
```

- 다음 행을 platform.ini 파일에 추가하십시오.
monitor _speed=115200
- 프로그램이 실행된 후 창 하단에 있는 버튼을 눌러 PlatformIO 터미널을 엽니다.



Lab 3:

RISC-V 어셈블리



RVfpga Lab 3: RISC-V 어셈블리

- RISC-V 어셈블리 언어 개요
- PlatformIO 프로젝트 만들기
- 프로젝트에 예제 RISC-V 어셈블리 프로그램 추가
- Nexys A7 보드에 RVfpgaNexys 다운로드
- RISC-V 어셈블리 프로그램을 RVfpga에 다운로드하고 프로그램 실행/디버그
- Lab 종료시 일부 또는 모든 연습 완료
- Verilator (using **RVfpgaSim**) or **Whisper** 에 있는 프로그램을 시뮬레이션 할 수 있습니다.

RVfpga Lab 3: RISC-V 조립 지침

일반적인 RISC-V 조립 지침 / Pseudo 지침

RISC-V Assembly	Description	Operation
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	$s3 = s4 \wedge s5$
andi t1, t2, 0xFFB	Bit-wise AND immediate	$t1 = t2 \& 0xFFFFFBB$
ori t0, t1, 0x2C	Bit-wise OR immediate	$t0 = t1 0x2C$
xori s3, s4, 0xABC	Bit-wise XOR immediate	$s3 = s4 \wedge 0xFFFFFABC$
sll t0, t1, t2	Shift left logical	$t0 = t1 \ll t2$
srl t0, t1, t5	Shift right logical	$t0 = t1 \gg t5$
sra s3, s4, s5	Shift right arithmetic	$s3 = s4 \ggg s5$
slli t1, t2, 30	Shift left logical immediate	$t1 = t2 \ll 30$
srli t0, t1, 5	Shift right logical immediate	$t0 = t1 \gg 5$
srai s3, s4, 31	Shift right arithmetic immediate	$s3 = s4 \ggg 31$

RVfpga Lab 3: RISC-V 어셈블리 설명서

일반적인 RISC-V 조립 지침 / Pseudo 지침 (계속)

RISC-V Assembly	Description	Operation
lw s7, 0x2C(t1)	Load word	s7 = memory[t1+0x2C]
lh s5, 0x5A(s3)	Load half-word	s5 = SignExt(memory[s3+0x5A] _{15:0})
lb s1, -3(t4)	Load byte	s1 = SignExt(memory[t4-3] _{7:0})
sw t2, 0x7C(t1)	Store word	memory[t1+0x7C] = t2
sh t3, 22(s3)	Store half-word	memory[s3+22] _{15:0} = t3 _{15:0}
sb t4, 5(s4)	Store byte	memory[s4+5] _{7:0} = t4 _{7:0}
beq s1, s2, L1	Branch if equal	if (s1==s2), PC = L1
bne t3, t4, Loop	Branch if not equal	if (s1!=s2), PC = Loop
blt t4, t5, L3	Branch if less than	if (t4 < t5), PC = L3
bge s8, s9, Done	Branch if not equal	if (s8>=s9), PC = Done
li s1, 0xABCDEF12	Load immediate	s1 = 0xABCDEF12
la s1, A	Load address	s1 = Memory address where variable A is stored
nop	Nop	no operation
mv s3, s7	Move	s3 = s7
not t1, t2	Not (Invert)	t1 = ~t2
neg s1, s3	Negate	s1 = -s3
j Label	Jump	PC = Label
jal L7	Jump and link	PC = L7; ra = PC + 4
jr s1	Jump register	PC = s1

RVfpga Lab 3: RISC-V 레지스터

32 비트 레지스터 32 개

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary variables
s0/fp	x8	Saved variable / Frame pointer
s1	x9	Saved variable
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved variables
t3-6	x28-31	Temporary variables

RVfpga Lab 3: RISC-V 어셈블리 프로그램의 예

```
• // memory-mapped I/O addresses
• # GPIO_SWs      = 0x80001400
• # GPIO_LEDs     = 0x80001404
• # GPIO_INOUT    = 0x80001408
•
• .globl main
• main:
•
• main:
•     li t0, 0x80001400    # base address of GPIO memory-mapped registers
•     li t1, 0xFFFF       # set direction of GPIOs
•                          # upper half = switches (inputs)    (=0)
•                          # lower half = outputs (LEDs)       (=1)
•     sw t1, 8(t0)        # GPIO_INOUT = 0xFFFF
•
• repeat:
•     lw  t1, 0(t0)        # read switches: t1 = GPIO_SWs
•     srli t1, t1, 16       # shift val to the right by 16 bits
•     sw  t1, 4(t0)        # write value to LEDs: GPIO_LEDs = t1
•     j   repeat          # repeat loop
```

이 프로그램은 스위치의 값을 LED에 기록합니다.



Lab 4:

함수 호출



RVfpga Lab 4: 함수 호출

- 함수 호출로 C 프로그램 작성
 - 함수는 프로시저라고도 합니다.
 -
- C 라이브러리 사용
- RISC-V (절차) 호출 규칙

RVfpga Lab 4: 함수가 있는 예제 프로그램

```
// 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 Lab 4: 함수가 있는 예제 프로그램

```
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 Lab 4: C 라이브러리

- 라이브러리
 - 일반적으로 사용되는 기능 모음
 - 공통 기능을 쉽게 사용할 수 있도록 제공 (프로그래밍 시간 절약)
- C 라이브러리 예제:
 - math.h (math 라이브러리): sqrt (제곱근), cos (코사인) 등과 같은 함수를 포함합니다.
 - stdio.h (표준 I/O 라이브러리): 화면에 값 출력 (printf), 사용자로부터 값 읽기 (scanf) 등의 기능을 포함합니다.
 - stdlib.h (표준 라이브러리): 난수 (rand)를 생성하는 함수를 포함합니다.
 - 기타 등등... (구글 C 라이브러리)

RVtpga Lab 4: C 라이브러리들 사용한 예제 프로그램

```
#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);  
}
```

이 프로그램은 0에서 65535 사이의 임의의 숫자를 LED에 씁니다.

RVfpga Lab 4: RISC-V 호출 규칙

- 함수 호출

`jal function_label`

- 함수에서 반환

`jr ra`

- 인수

- 레지스터에 배치 `a0-a7`

- 반환 값

- 레지스터에 배치 `a0`

RVfpga Lab 4: RISC-V 호출 규칙 예

C 코드

```
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 어셈블리

```
# 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 Lab 4: 스택

- 레지스터 값을 저장하는 데 사용되는 메모리의 스크래치 공간
- 스택 포인터 (sp)는 스택 맨 위의 주소를 보유합니다.
- 스택은 메모리에서 아래쪽으로 커집니다. 예를 들어 스택에 4 단어 (16 바이트)를 위한 공간을 만들기 위해 다음 코드가 사용됩니다.

```
addi sp, sp, -16
```

- 두 가지 범주의 레지스터:
 - **보존된 레지스터:** 레지스터 내용은 함수 호출간에 **보존**되어야 합니다 (즉, 함수 호출 전후에 동일한 값을 포함).
 - **보존되지 않은 레지스터:** 레지스터 내용은 함수 호출간에 **보존**되지 않아야 합니다 (즉, 레지스터가 함수 호출 전후에 동일할 필요는 없음).
 - 저장된 레지스터 ($s0-s11$), 반환 주소 레지스터 (ra) 및 스택 포인터 (sp)는 **보존**된 레지스터입니다. 다른 모든 레지스터는 **보존**되지 않습니다.

RVfpga Lab 4: 보존 / 비 보존 레지스터

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 Lab 4: 스택 – 수정 된 어셈블리 코드

C 코드

```
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 어셈블리

```
# 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
```

Lab 5:

C 및 어셈블리



RVfpga Lab 5: C와 어셈블리 결합

- 예: 이미지 처리 프로그램
- C로 작성된 일부 함수와 어셈블리로 작성된 일부 함수

RVfpga Lab 5: 이미지 처리 프로그램

- 컬러 이미지를 그레이 스케일로 변환



RVfpga Lab 5: 이미지 처리 프로그램

- 3 개의 8 비트 색상으로 저장된 각 픽셀: **R** = 빨간색, **G** = 녹색, **B** = 파란색
- R, G 및 B 값을 변경하여 모든 색상을 만들 수 있습니다.
- 이미지를 8 비트 회색조 (**grey**)로 변환하기 위해 각 픽셀은 다음과 같이 변환됩니다:

$$\text{grey} = (306 * \text{R} + 601 * \text{G} + 117 * \text{B}) >> 10$$

- RGB 가중치의 합은 1024 ($306 + 601 + 117 = 1024$)가 되므로 8 비트 범위 (0-255)로 돌아가려면 결과를 1024로 나눕니다. (즉, 10 비트 오른쪽으로 이동: $>> 10$)
- 알고리즘에 대한 자세한 내용은:

<https://www.mathworks.com/help/matlab/ref/rgb2gray.html>

RVfpga Lab 5: 어셈블리 기능

`.globl ColourToGrey_Pixel` ← `.globl`은 `ColourToGrey_Pixel` 기능을 프로젝트의 모든 파일에 표시합니다.

`ColourToGrey_Pixel:`

```
li x28, 306          # a0 = R * 306
mul a0, a0, x28
li x28, 601          # a1 = G * 601
mul a1, a1, x28
li x28, 117          # a2 = B * 117
mul a2, a2, x28
add a0, a0, a1        # grey = a0 + a1 + a2
add a0, a0, a2
srl a0, a0, 10        # grey = grey / 1024
ret                  # return
.end
```

$$\text{grey} = (306 * \text{R} + 601 * \text{G} + 117 * \text{B}) \gg 10$$

RVfpga Lab 5: 구조체와 배열

```
typedef struct {
    unsigned char R;
    unsigned char G;
    unsigned char B;
} RGB;

extern unsigned char VanGogh_128x128[]; // 1D array of individual RGB values
RGB ColourImage[N][M];                // 2D array of RGB struct (colour image)
unsigned char GreyImage[N][M];        // 2D array of greyscale image

// VanGogh_128.c
unsigned char VanGogh_128x128[] = { 157, // R (pixel [0][0])
                                     182, // G (pixel [0][0])
                                     161, // B (pixel [0][0])
                                     171, // R (pixel [0][1])
                                     195, // G (pixel [0][1])
                                     173, // B (pixel [0][1])
                                     173, // R (pixel [0][2])
                                     ... }
```


RVfpga Lab 5: 주요 기능

```
int main(void) {
    // Create an N x M matrix using the input image
    initColourImage(ColourImage);

    // Transform Colour Image to Grey Image
    ColourToGrey(ColourImage, GreyImage);
    ...
}

void ColourToGrey(RGB Colour[N][M], unsigned char Grey[N][M]) {
    int i, j;

    for (i=0; i<N; i++)
        for (j=0; j<M; j++)
            Grey[i][j] = ColourToGrey_Pixel(Colour[i][j].R, Colour[i][j].G,
                                                Colour[i][j].B);
}
```

Lab 6:

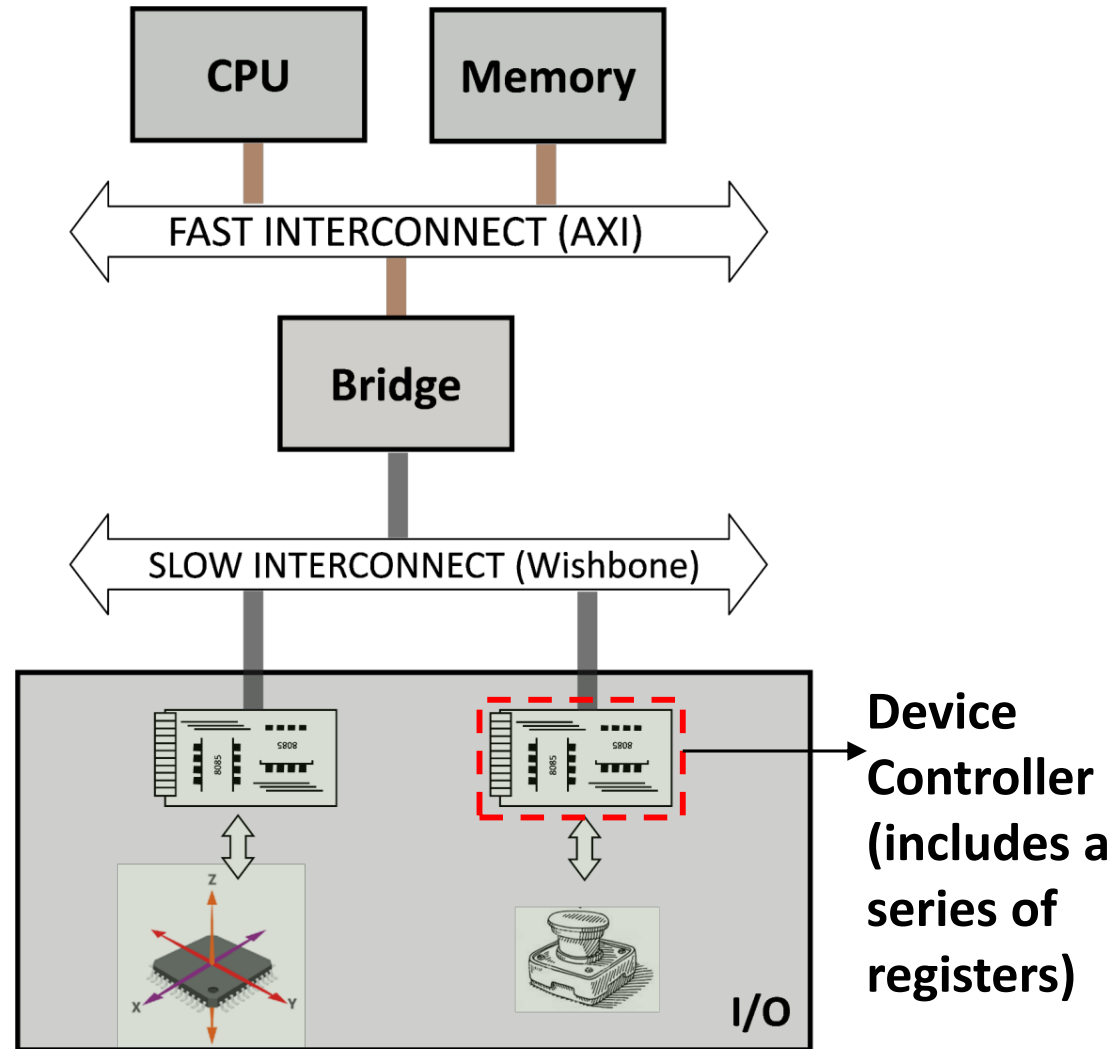
I/O 소개



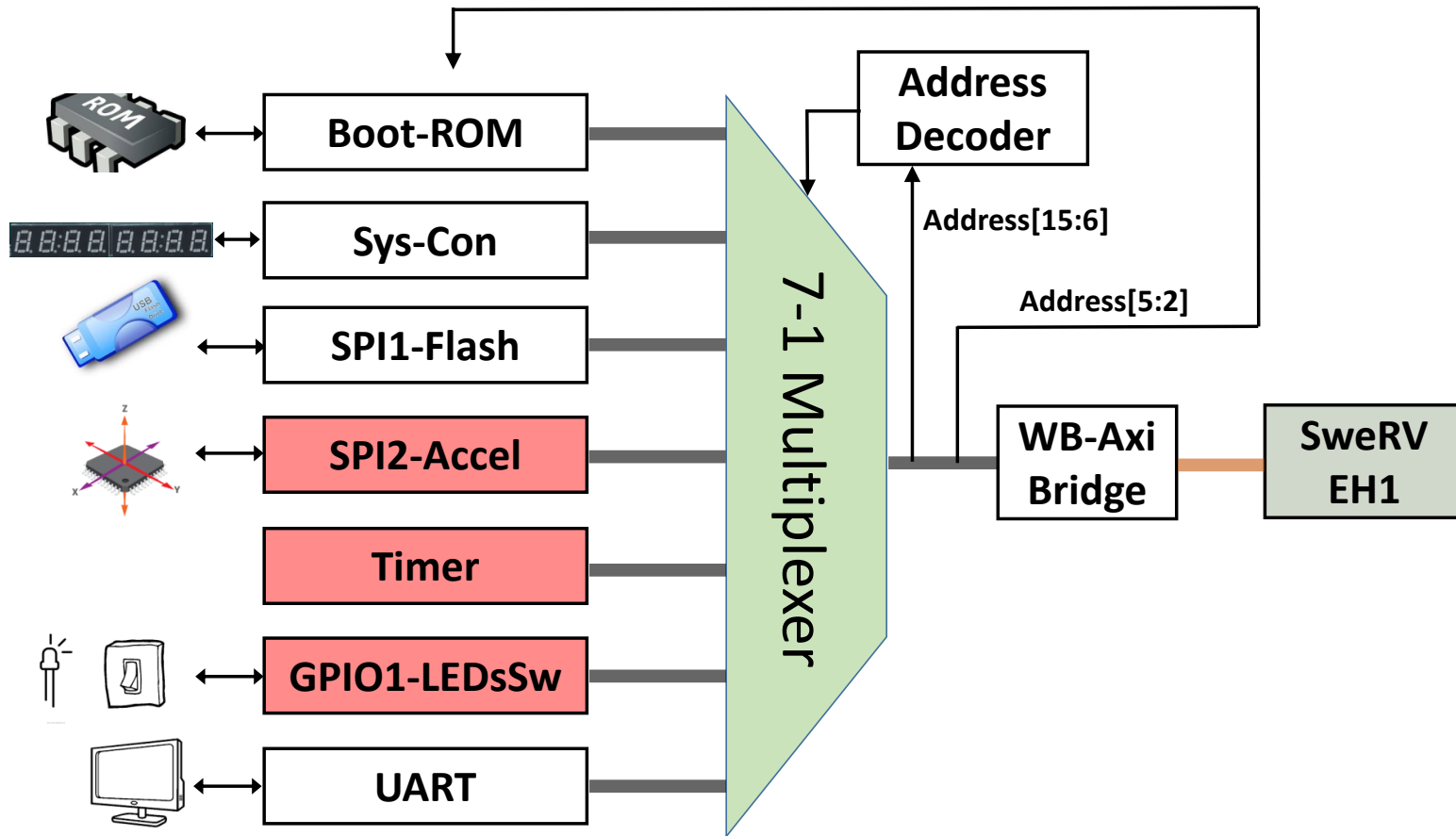
RVfpga Lab 6: I/O 소개

- 입출력 (I/O) 시스템 – 주변 장치라고도 함
- 범용 I/O (GPIO)
- GPIO 컨트롤러

RVfpga Lab 6: I/O가있는 일반 프로세서



RVfpga Lab 6: I/O가있는 프로세서



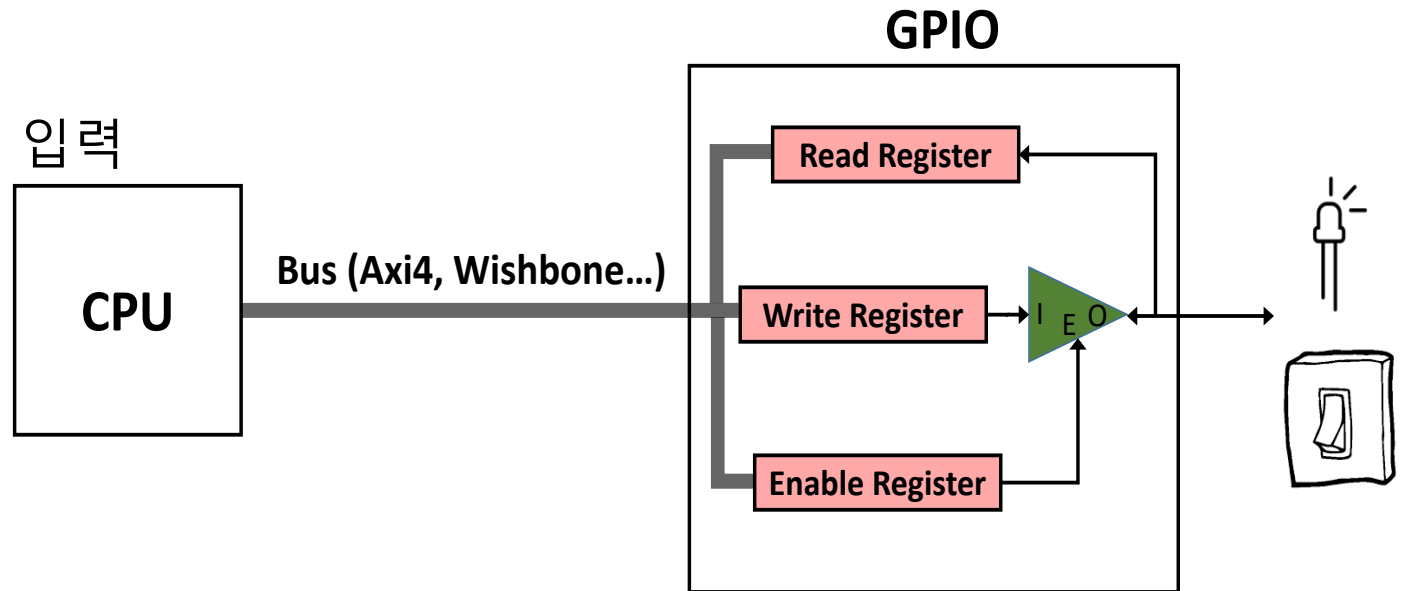
Peripherals

- SweRVofX 주변 장치:
 - 부팅 ROM
 - 시스템 컨트롤러
 - SPI1 플래시 메모리
 - UART
 - GPIO LED 및 스위치
 - 타이머
 - SPI2 가속도계
 - 7 세그먼트 디스플레이 (시스템 컨트롤러: Sys-Con)

RVfpga Lab 6: 범용 I/O (GPIO)

- 범용 I/O:
 - 프로세서가 주변 장치 (예: 스위치 및 LED)에 연결된 핀을 읽고 쓸 수 있습니다.
 - 각 핀은 3 상태를 사용하여 입력 또는 출력으로 구성할 수 있습니다.
- 3 개의 메모리 매핑 레지스터:
 - 읽기 레지스터: 핀에서 읽은 값
 - 쓰기 레지스터: 핀에 쓸 값
 - 레지스터 활성화: 1 = 출력, 0 = 입력

주변기기



RVfpga Lab 6: 메모리 매핑 레지스터

레지스터	메모리 매핑 주소
레지스터 읽기	0x80001400
레지스터 쓰기	0x80001404
등록 활성화	0x80001408

- GPIO의 비트 15:0을 출력으로, 31:16을 입력으로 구성:

```
li t0, 0x80001400 # t0 = 0x80001400
```

```
li t1, 0xFFFF # 1 = output, 0 = input
```

```
sw t1, 8(t0) # [15:0] = outputs, [31:16] = inputs
```

- I/O 읽기:

```
lw t2, 0(t0) # t2 = value of GPIO pins
```

- I/O 쓰기:

```
sw t3, 4(t0) # GPIO pins = t3
```

RVfpga Lab 6: RVfpga GPIO 모듈

- OpenCores의 GPIO 모듈

<https://opencores.org/projects/gpio>

- 최대 32 개의 GPIO 핀 허용

- 모든 핀은 개별적으로 입력 (활성화 = 0) 또는 출력 (활성화 = 1)으로 구성할 수 있습니다.
- 프로그램 전체에 걸쳐 구성 변경 가능

레지스터	메모리 매핑 주소
레지스터 읽기	0x80001400
레지스터 쓰기	0x80001404
등록 활성화	0x80001408

RVfpga Lab 6: 메모리 매핑 레지스터

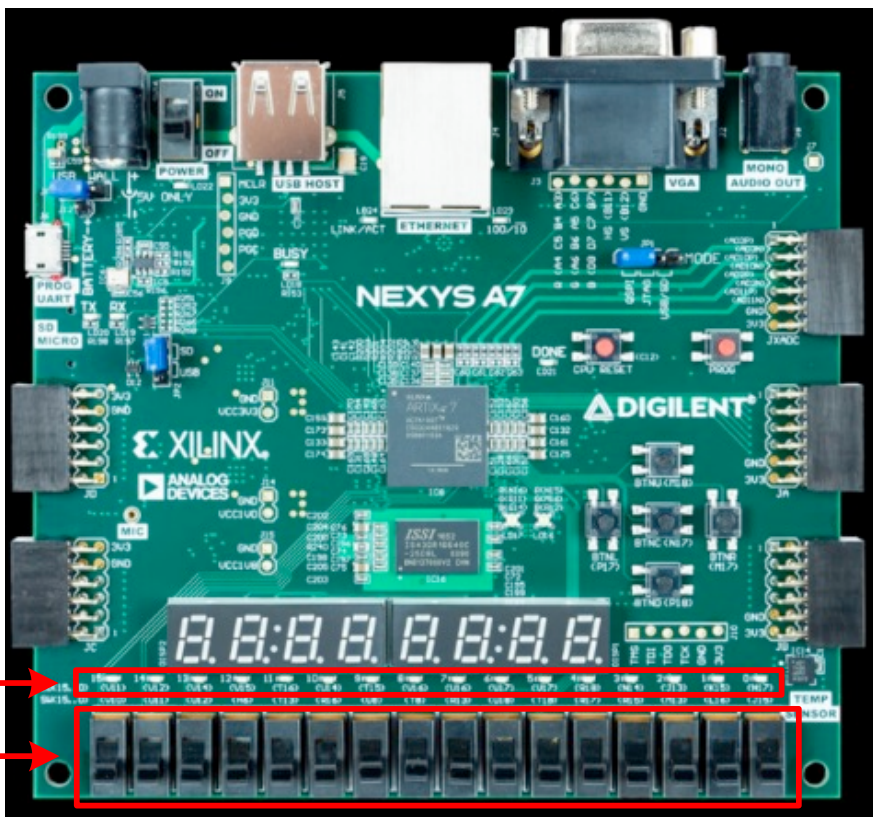


figure of board from <https://reference.digilentinc.com/>

LED 및 스위치를 GPIO 핀에 매핑:

- LED: 핀 **[15:0]** (프로세서 출력)
- 스위치: 핀 **[31:16]** (프로세서에 대한 입력)

GPIO 구성:

- 레지스터 활성화 = 0x0000FFFF (1 = 출력, 0 = 입력)

```
li t0, 0x80001400
li t1, 0xFFFF
sw t1, 8(t0) # Enable Register = 0xFFFF
```

LED 쓰기 :

- [15: 0]의 값을 0x80001404 주소에 씁니다.
`sw t3, 4(t0) # LEDs = [t3]15:0`

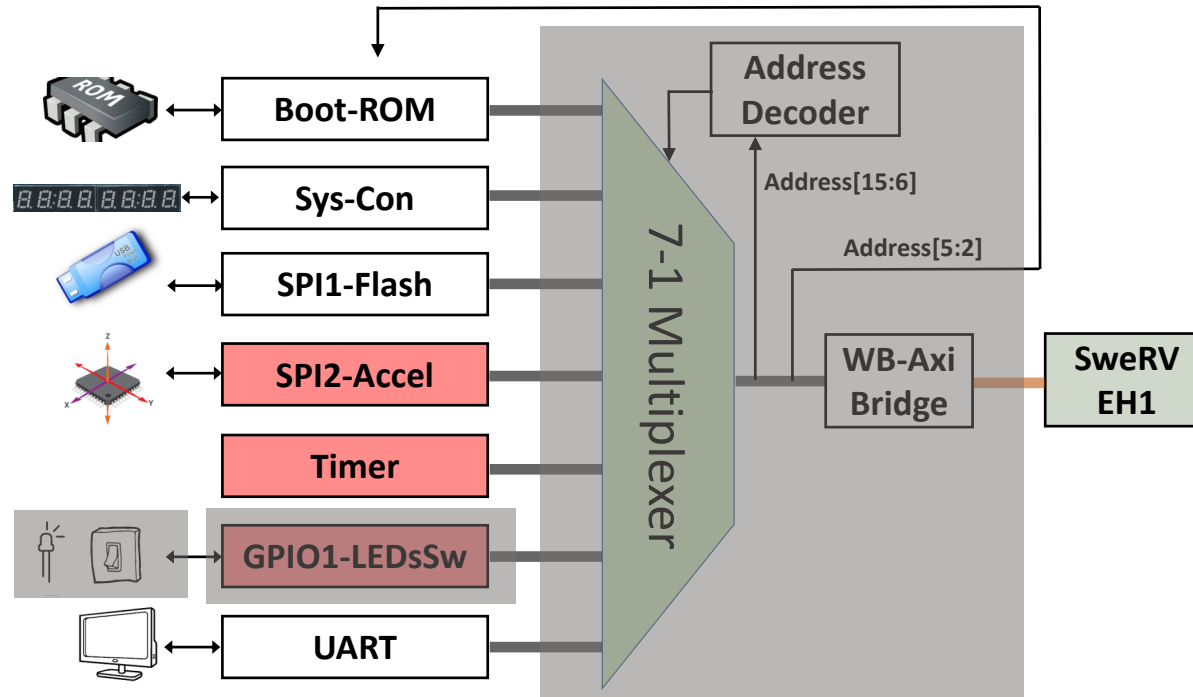
읽기 스위치:

- 주소 0x80001400에서 비트 [31:16]의 읽기 스위치
- 16 비트만큼 오른쪽으로 이동하여 하위 16 비트에 값을 넣습니다.


```
lw    t5, 0(t0)    # [t5]31:16 = switch values
srli  t5, t5, 16    # [t5]15:0 = switch values
```

RVfpga Lab 6: GPIO 초급 수준 구현

- 3 개의 주요 부분으로 나뉩니다.
 - 온보드 LED/스위치에 대한 RVfpgaNexys의 외부 연결 (왼쪽 음영 영역)
 - GPIO 모듈을 SweRVolfX에 통합 (중간 음영 영역)
 - GPIO와 SweRV EH1 (오른쪽 음영 영역) 간의 연결



RVfpga Lab 6: 외부 연결

파일 rvfpganexys.xdc: 온보드 스위치가 있는 i_sw [15: 0] 및 온보드 LED가 있는 o_led [15: 0]의 연결을 정의합니다.

```
26 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[0] }]
27 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[1] }]
28 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[2] }]
29 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[3] }]
30 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { i_sw[4] }]
31 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[5] }]
32 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[6] }]
33 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[7] }]
34 set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[8] }]
35 set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[9] }]
36 set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[10] }]
37 set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[11] }]
38 set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { i_sw[12] }]
39 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { i_sw[13] }]
40 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { i_sw[14] }]
41 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { i_sw[15] }]
42
43 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
44 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { o_led[1] }]
45 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { o_led[2] }]
46 set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { o_led[3] }]
47 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { o_led[4] }]
48 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { o_led[5] }]
49 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { o_led[6] }]
50 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { o_led[7] }]
51 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { o_led[8] }]
52 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { o_led[9] }]
53 set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { o_led[10] }]
54 set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { o_led[11] }]
55 set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { o_led[12] }]
56 set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { o_led[13] }]
57 set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { o_led[14] }]
58 set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { o_led[15] }]
```


RVfpga Lab 6: RVfpga에 통합

파일 swervolf_core.v: Tri-state 버퍼 및 GPIO 모듈 인스턴스화

```
bidirec gpio0 (.oe(en_gpio[0]), .inp(o_gpio[0]), .outp(i_gpio[0]), .bidir(io_data[0]));
bidirec gpio1 (.oe(en_gpio[1]), .inp(o_gpio[1]), .outp(i_gpio[1]), .bidir(io_data[1]));
bidirec gpio2 (.oe(en_gpio[2]), .inp(o_gpio[2]), .outp(i_gpio[2]), .bidir(io_data[2]));
bidirec gpio3 (.oe(en_gpio[3]), .inp(o_gpio[3]), .outp(i_gpio[3]), .bidir(io_data[3]));
bidirec gpio4 (.oe(en_gpio[4]), .inp(o_gpio[4]), .outp(i_gpio[4]), .bidir(io_data[4]));
bidirec gpio5 (.oe(en_gpio[5]), .inp(o_gpio[5]), .outp(i_gpio[5]), .bidir(io_data[5]));
bidirec gpio6 (.oe(en_gpio[6]), .inp(o_gpio[6]), .outp(i_gpio[6]), .bidir(io_data[6]));
bidirec gpio7 (.oe(en_gpio[7]), .inp(o_gpio[7]), .outp(i_gpio[7]), .bidir(io_data[7]));
bidirec gpio8 (.oe(en_gpio[8]), .inp(o_gpio[8]), .outp(i_gpio[8]), .bidir(io_data[8]));
bidirec gpio9 (.oe(en_gpio[9]), .inp(o_gpio[9]), .outp(i_gpio[9]), .bidir(io_data[9]));
bidirec gpio10 (.oe(en_gpio[10]), .inp(o_gpio[10]), .outp(i_gpio[10]), .bidir(io_data[10]));
bidirec gpio11 (.oe(en_gpio[11]), .inp(o_gpio[11]), .outp(i_gpio[11]), .bidir(io_data[11]));
bidirec gpio12 (.oe(en_gpio[12]), .inp(o_gpio[12]), .outp(i_gpio[12]), .bidir(io_data[12]));
bidirec gpio13 (.oe(en_gpio[13]), .inp(o_gpio[13]), .outp(i_gpio[13]), .bidir(io_data[13]));
bidirec gpio14 (.oe(en_gpio[14]), .inp(o_gpio[14]), .outp(i_gpio[14]), .bidir(io_data[14]));
bidirec gpio15 (.oe(en_gpio[15]), .inp(o_gpio[15]), .outp(i_gpio[15]), .bidir(io_data[15]));
bidirec gpio16 (.oe(en_gpio[16]), .inp(o_gpio[16]), .outp(i_gpio[16]), .bidir(io_data[16]));
bidirec gpio17 (.oe(en_gpio[17]), .inp(o_gpio[17]), .outp(i_gpio[17]), .bidir(io_data[17]));
bidirec gpio18 (.oe(en_gpio[18]), .inp(o_gpio[18]), .outp(i_gpio[18]), .bidir(io_data[18]));
bidirec gpio19 (.oe(en_gpio[19]), .inp(o_gpio[19]), .outp(i_gpio[19]), .bidir(io_data[19]));
bidirec gpio20 (.oe(en_gpio[20]), .inp(o_gpio[20]), .outp(i_gpio[20]), .bidir(io_data[20]));
bidirec gpio21 (.oe(en_gpio[21]), .inp(o_gpio[21]), .outp(i_gpio[21]), .bidir(io_data[21]));
bidirec gpio22 (.oe(en_gpio[22]), .inp(o_gpio[22]), .outp(i_gpio[22]), .bidir(io_data[22]));
bidirec gpio23 (.oe(en_gpio[23]), .inp(o_gpio[23]), .outp(i_gpio[23]), .bidir(io_data[23]));
bidirec gpio24 (.oe(en_gpio[24]), .inp(o_gpio[24]), .outp(i_gpio[24]), .bidir(io_data[24]));
bidirec gpio25 (.oe(en_gpio[25]), .inp(o_gpio[25]), .outp(i_gpio[25]), .bidir(io_data[25]));
bidirec gpio26 (.oe(en_gpio[26]), .inp(o_gpio[26]), .outp(i_gpio[26]), .bidir(io_data[26]));
bidirec gpio27 (.oe(en_gpio[27]), .inp(o_gpio[27]), .outp(i_gpio[27]), .bidir(io_data[27]));
bidirec gpio28 (.oe(en_gpio[28]), .inp(o_gpio[28]), .outp(i_gpio[28]), .bidir(io_data[28]));
bidirec gpio29 (.oe(en_gpio[29]), .inp(o_gpio[29]), .outp(i_gpio[29]), .bidir(io_data[29]));
bidirec gpio30 (.oe(en_gpio[30]), .inp(o_gpio[30]), .outp(i_gpio[30]), .bidir(io_data[30]));
bidirec gpio31 (.oe(en_gpio[31]), .inp(o_gpio[31]), .outp(i_gpio[31]), .bidir(io_data[31]));
```

```
gpio_top gpio_module(
    .wb_clk_i      (clk),
    .wb_rst_i      (wb_rst),
    .wb_cyc_i      (wb_m2s_gpio_cyc),
    .wb_adr_i      ({2'b0,wb_m2s_gpio_adr[5:2],2'b0}),
    .wb_dat_i      (wb_m2s_gpio_dat),
    .wb_sel_i      (4'b1111),
    .wb_we_i       (wb_m2s_gpio_we),
    .wb_stb_i      (wb_m2s_gpio_stb),
    .wb_dat_o      (wb_s2m_gpio_dat),
    .wb_ack_o      (wb_s2m_gpio_ack),
    .wb_err_o      (wb_s2m_gpio_err),
    .wb_inta_o     (gpio_irq),
    // External GPIO Interface
    .ext_pad_i     (i_gpio[31:0]),
    .ext_pad_o     (o_gpio[31:0]),
    .ext_padoe_o   (en_gpio));
```


RVfpga Lab 6: SweRV EH1과 연결

파일 wb_intercon.v: 7-1 Multiplexer 구현

```
108 wb_mux
109 #(.num_slaves (7),
110   .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00002000}),
111   .MATCH_MASK ({32'hfffff000, 32'hfffffc0, 32'hfffffc0, 32'hfffffc0, 32'hfffffc0, 32'hfffffc0, 32'hffff000}))
112 wb_mux_io
113 (.wb_clk_i (wb_clk_i),
114  .wb_rst_i (wb_rst_i),
115  .wbm_adr_i (wb_io_adr_i),
116  .wbm_dat_i (wb_io_dat_i),
117  .wbm_sel_i (wb_io_sel_i),
118  .wbm_we_i (wb_io_we_i),
119  .wbm_cyc_i (wb_io_cyc_i),
120  .wbm_stb_i (wb_io_stb_i),
121  .wbm_cti_i (wb_io_cti_i),
122  .wbm_bte_i (wb_io_bte_i),
123  .wbm_dat_o (wb_io_dat_o),
124  .wbm_ack_o (wb_io_ack_o),
125  .wbm_err_o (wb_io_err_o),
126  .wbm_rty_o (wb_io_rty_o),
127  .wbs_adr_o ({wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o}),
128  .wbs_dat_o ({wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o}),
129  .wbs_sel_o ({wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o}),
130  .wbs_we_o ({wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_uart_we_o}),
131  .wbs_cyc_o ({wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o}),
132  .wbs_stb_o ({wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o}),
133  .wbs_cti_o ({wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o}),
134  .wbs_bte_o ({wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o}),
135  .wbs_dat_i ({wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i}),
136  .wbs_ack_i ({wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i}),
137  .wbs_err_i ({wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i}),
138  .wbs_rty_i ({wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i});
139
140 endmodule
```

CPU/Controller Signals

Peripheral Signals



Lab 7:

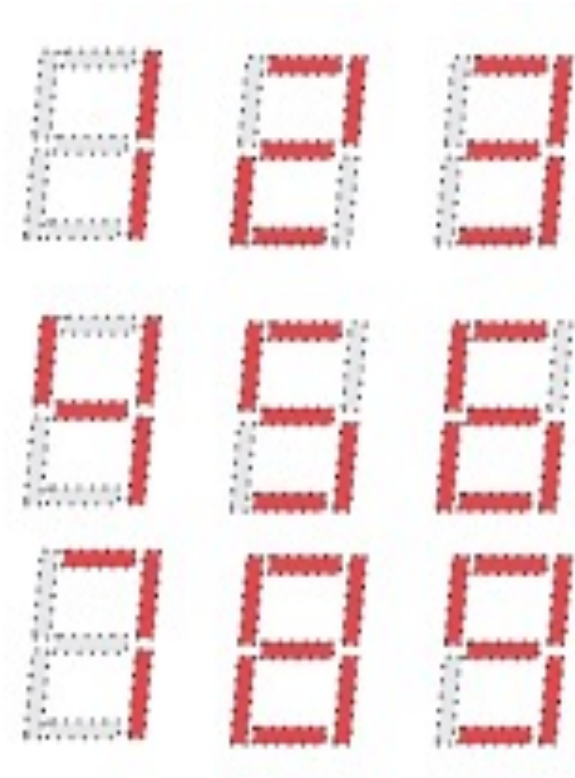
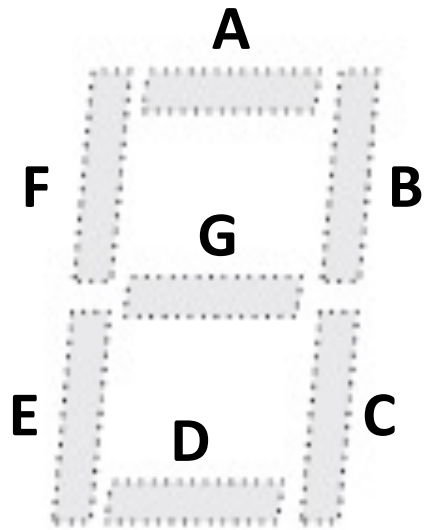
7 세그먼트 디스플레이



RVfpga Lab 7: 7 세그먼트 디스플레이

- 7 세그먼트 디스플레이 개요
- 7 세그먼트 디스플레이 하드웨어

RVfpga Lab 7: 7 세그먼트 디스플레이 개요



- 7 개의 LED 세그먼트: A-G
- 특정 숫자를 만들기 위한 세그먼트 조명
 - 1: 세그먼트 B 및 C
 - 2: 세그먼트 A, B, D, E, G
 - 3: 세그먼트 A, B, C, D, G
 - 기타

RVtpga Lab 7: Nexys A7의 7 세그먼트 디스플레이

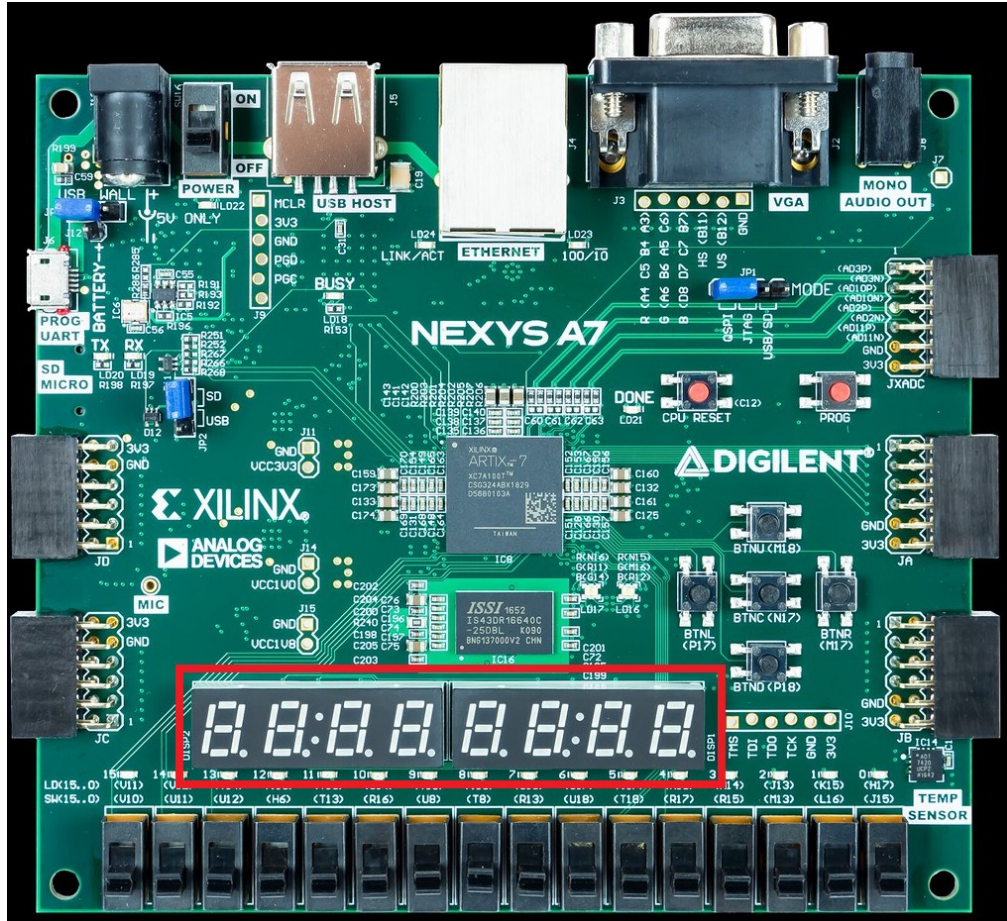


figure of board from <https://reference.digilentinc.com/>

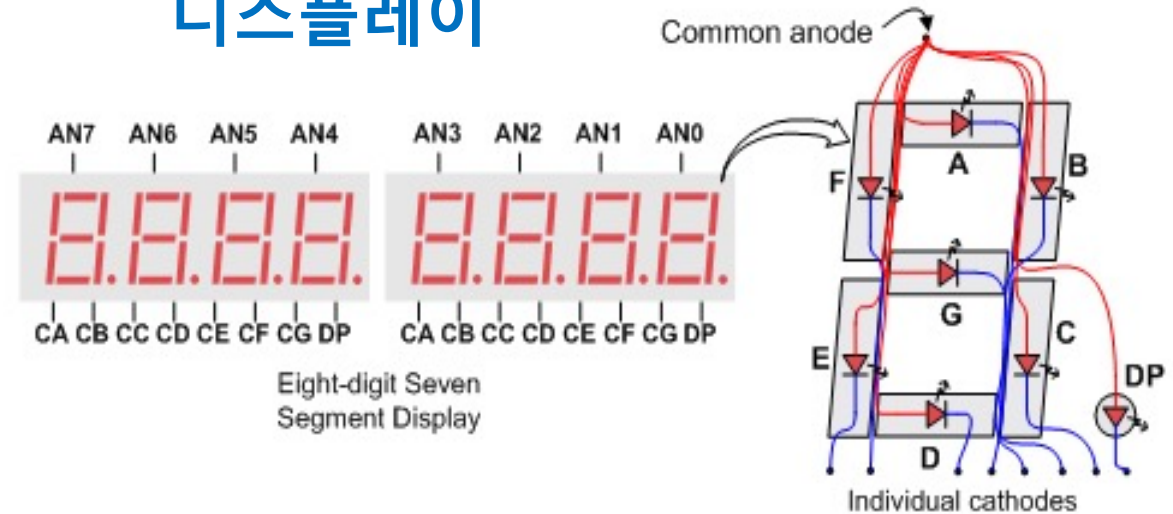
- 8 자리 7 세그먼트 디스플레이
- 메모리 매핑 액세스:
 - Enables_Reg: 0x80001038
 - Digits_Reg: 0x8000103C
- Enables는 **low-asserted** 입니다
- 예: 맨 오른쪽 두 자리에 71 표시:
 - Enables_Reg = 0xFC (0b11111100: 맨 오른쪽 두 자리 활성화)
 - Digits_Reg = 0x71
 - 어셈블리:

```
li t0, 0x80001038
li t1, 0xFC
li t2, 0x71
sw t1, 0(t0)
sw t2, 4(t0)
```

RVfpga Lab 7: 7 세그먼트 디스플레이 하드웨어

- 각 숫자는 공통 양극 (해당 숫자 LED의 양극이 서로 연결됨)
 - 양극 신호가 활성화 (AN0-AN7) 역할을 합니다.
 - 숫자를 활성화하려면 low 드라이브 (AN0-AN7은 LED에 공급되기 전에 인버터 (표시되지 않음)를 통과 함)
- 모든 숫자에 대한 각 세그먼트는 함께 묶여 있음
 - 세그먼트를 켜기 위해 low로 구동됩니다.
 - AN0-AN7 신호의 **Time-multiplexing**를 통해 각 숫자에 고유한 값을 표시할 수 있습니다.
 - 숫자의 AN 신호 (AN0-AN7)는 1-16ms마다 작아야 밝아집니다.

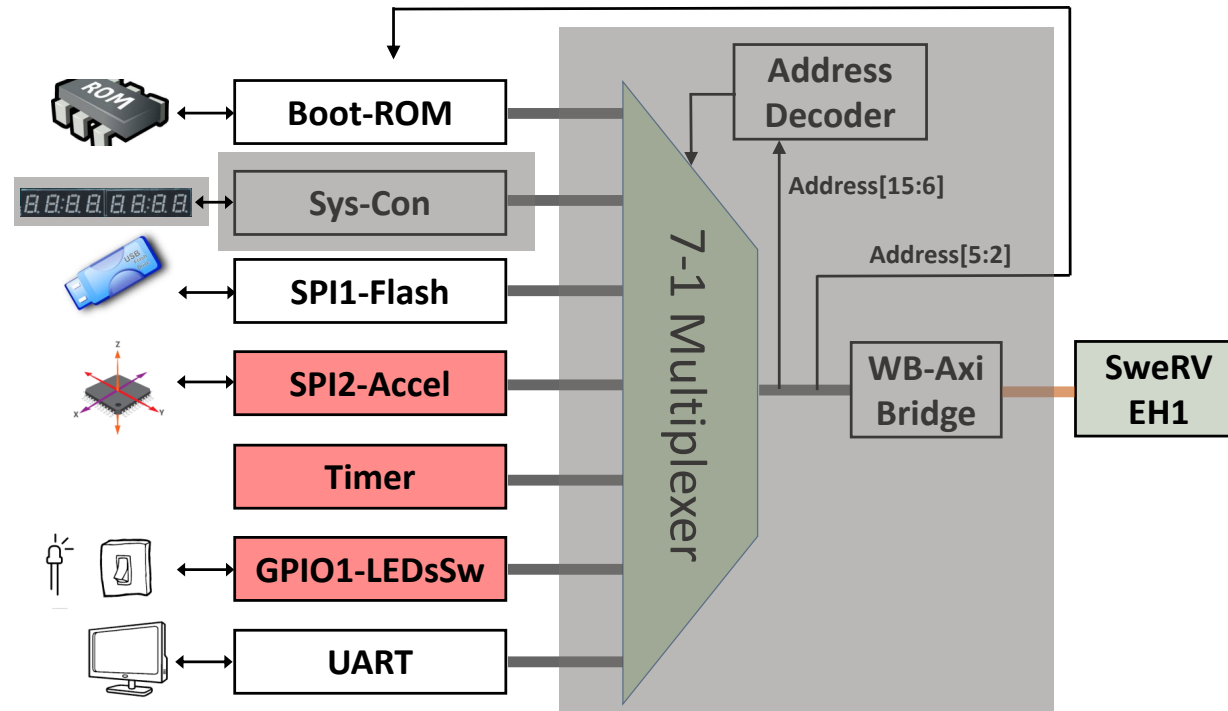
8 자리 7 세그먼트 디스플레이



RVfpga Lab 7: 7 세그먼트Disp. 초급 수준의 구현

- 3 개의 주요 부분으로 나뉩니다.

- 온보드 7 세그먼트에 대한 RVfpgaNexys의 외부 연결. 디스플레이 (왼쪽 음영 영역)
- 7 세그먼트 디스플레이 통합 모듈을 SweRVolfX (중간 음영 영역) 안으로 구현
- 7 세그먼트 디스플레이와 SweRV EH1 연결 (오른쪽 음영 영역)



RVfpga Lab 7: 외부 연결

파일 rvfpganexys.xdc: 온보드 7 세그먼트 디스플레이와 CA-CG (SoC에서 Digits_Bits [i]라고 함) 및 AN [i]의 연결을 정의합니다.

```
60 ##7 segment display
61 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { CA }]; #IO_L24N_T3_A00_D16_14 Sch=ca
62 set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { CB }]; #IO_25_14 Sch=cb
63 set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { CC }]; #IO_25_15 Sch=cc
64 set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { CD }]; #IO_L17P_T2_A26_15 Sch=cd
65 set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { CE }]; #IO_L13P_T2_MRCC_14 Sch=ce
66 set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { CF }]; #IO_L19P_T3_A10_D26_14 Sch=cf
67 set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { CG }]; #IO_L4P_T0_D04_14 Sch=cg
68 #set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
69 set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
70 set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
71 set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
72 set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
73 set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
74 set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
75 set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
76 set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
77
```

RVfpga Lab 7: SweRVolfX에 통합

- 파일 swervolf_syscon.v: 7세그먼트는 컨트롤러 인스턴스화를 표시합니다. 모듈은 클럭 신호 (i_clk) 및 리셋 신호 (i_rst) 외에도 앞에서 설명한 두 개의 메모리 매핑 제어 레지스터인 두 개의 입력 신호 Enables_Reg 및 Digits_Reg를 수신합니다. 이 모듈은 보드의 7 세그먼트 디스플레이에 연결된 두 개의 신호 AN 및 Digits_Bits를 출력합니다.

```
// Eight-Digit 7 Segment Displays

reg [ 7:0] Enables_Reg;
reg [31:0] Digits_Reg;

SevSegDisplays_Controller SegDispl_Ctr(
    .clk          (i_clk),
    .rst_n        (i_rst),
    .Enables_Reg  (Enables_Reg),
    .Digits_Reg   (Digits_Reg),
    .AN           (AN),
    .Digits_Bits  (Digits_Bits)
);
```

RVfpga Lab 7: SweRVolfX에 통합

- SevSegDisplays_Controller도 이 파일에서 구현됩니다. 여기에는 다음과 같은 하위 단위가 포함됩니다.
 - 2ms마다 AN 및 Digits_Bits 신호로 보낼 값을 선택하는 두 개의 멀티플렉서 (모듈 SevSegMux).
 - 2ms 주기를 생성하는 카운터 (모듈 카운터).
 - 지정된 4 비트 16 진수 값에 대한 세그먼트 값을 출력하는 디코더 (모듈 SevenSegDecoder).

Lab 8:

타이머



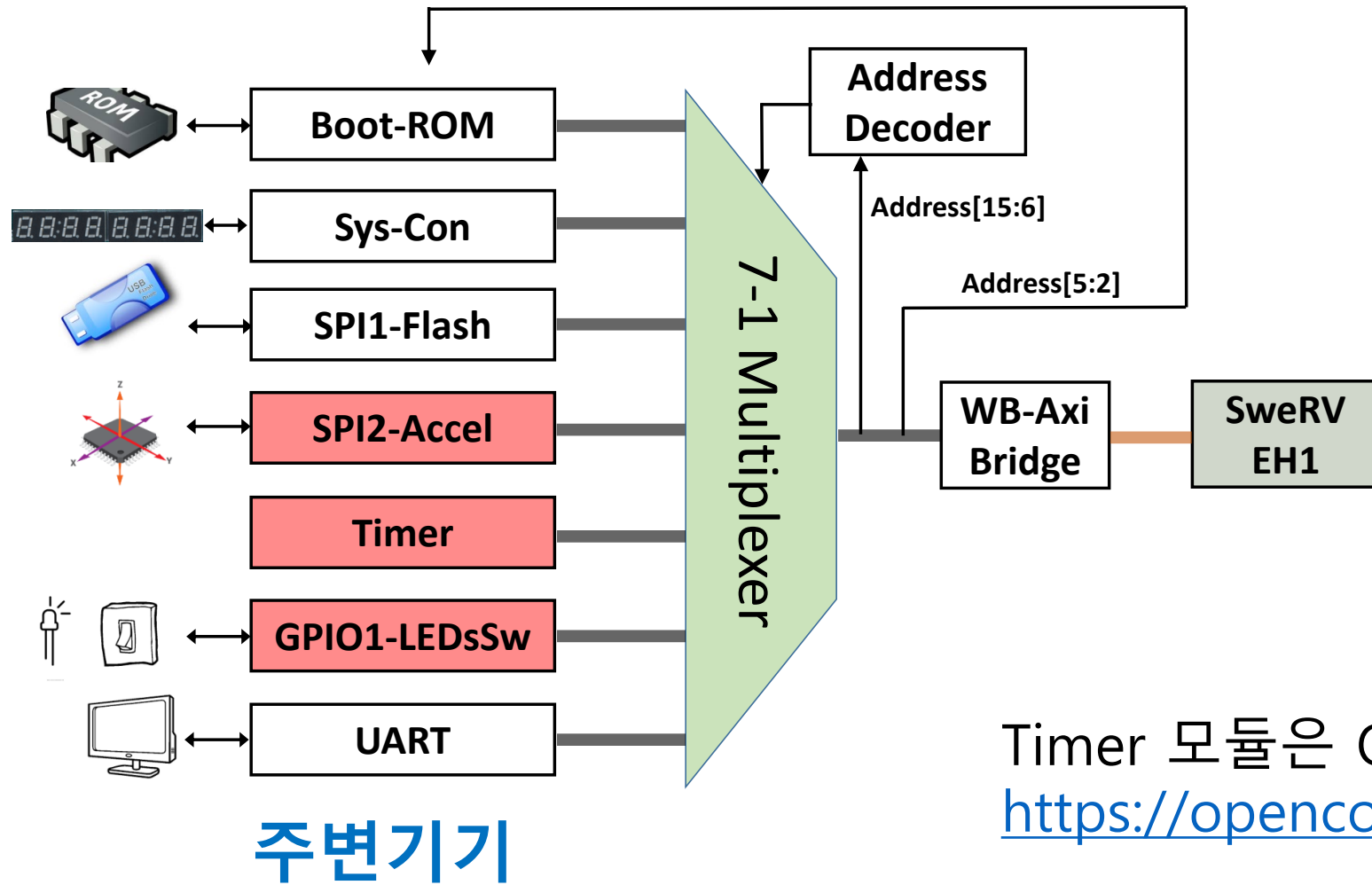
RVfpga Lab 8: 타이머

- 타이머 개요
- 타이머 레지스터
- 타이머 예제

RVfpga Lab 8: 타이머

- 타이머는 고정된 주파수로 카운터를 증가 또는 감소시킵니다.
- 마이크로 컨트롤러 및 SoC에서 일반적으로 발견
- 정확한 타이밍 생성에 사용

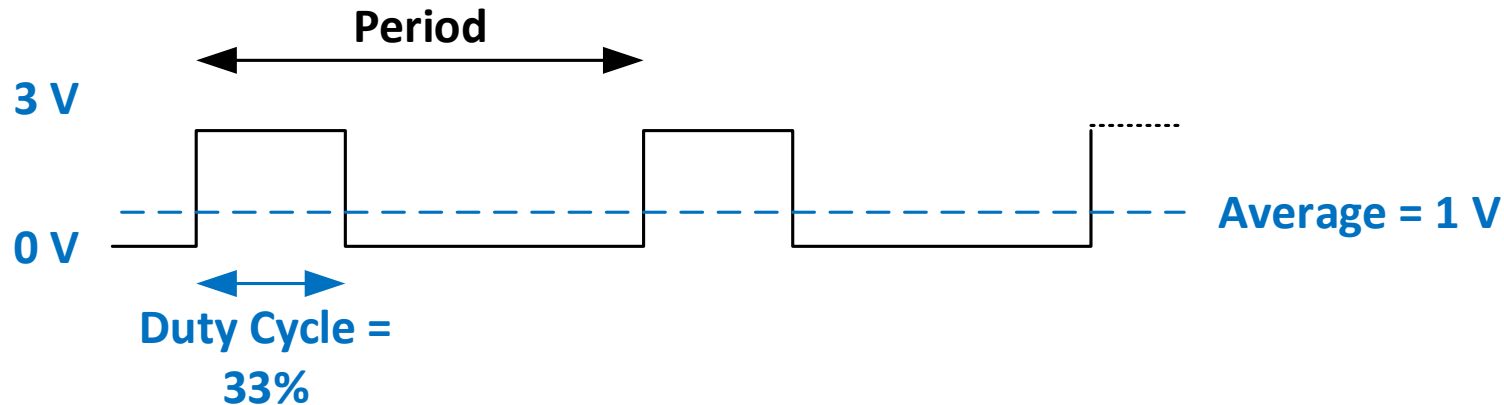
RVfpga Lab 8: 타이머



Timer 모듈은 OpenCores에서 제공합니다.
<https://opencores.org/projects/ptc>

RVfpga Lab 8: 타이머 (PTC) 모듈

- 타이머 모듈 (PTC 모듈이라고도 함)은 다음에 사용됩니다.
 - 타이머(T) / 카운터(C): 클럭 에지 (또는 이벤트라고도 하는 다른 신호의 에지)를 계산합니다.
 - 펄스(P) 폭 변조 (PWM):
 - 출력의 높은 지속 시간 (듀티 사이클이라고 함) 변경
 - 아날로그 전압을 디지털 방식으로 근사화하는데 사용
- PWM 예:** 33 % 듀티 사이클 (신호는 시간의 1/3이 높음). 하이 레벨이 3V이면 아날로그 전압 (신호의 평균 전압)은 $3V * 0.33 = 1V$ 입니다.



RVfpga Lab 8: 타이머 (PTC) 레지스터

Name	Address	Width	Access	Description
RPTC_CNTR	0x80001200	1-32	R/W	Main PTC counter
RPTC_HRC	0x80001204	1-32	R/W	PTC HI Reference/Capture register
RPTC_LRC	0x80001208	1-32	R/W	PTC LO Reference/Capture register
RPTC_CTRL	0x8000120C	9	R/W	Control register

- **RPTC_CNTR**: 카운터 (카운터 값)
- **RPTC_HRC**: 하이 레퍼런스 캡처(High reference capture) – PWM 모드에서 출력이 high로 전환되어야 하는 사이클 수 (리셋 후)를 나타냅니다.
- **RPTC_LRC**: 낮은 레퍼런스 캡처(Low reference capture) – 카운터/타이머 모드에서 카운트가 완료되었을 때 (리셋 후) 사이클 수를 나타냅니다. PWM 모드에서 출력이 낮아야 할 때 (리셋 후) 클럭 사이클 수를 나타냅니다.
- **RPTC_CTRL**: 제어 레지스터

RVfpga Lab 8: 타이머 (PTC) 제어 레지스터

Bit	Access	Reset	Name & Description
0	R/W	0	EN: 설정시 RPTC_CNTR이 증가합니다..
1	R/W	0	ECLK: 클럭 신호를 선택합니다: 외부 클럭, ptc_ecgt (1) 또는 시스템 클럭 (0)을 통해.
2	R/W	0	NEC: 외부 클럭 (ptc_ecgt)의 네거티브/포지티브 에지 및 low/high 기간을 선택하는 데 사용됩니다.
3	R/W	0	OE: PWM 출력 드라이버를 활성화합니다.
4	R/W	0	SINGLE: 설정된 경우 RPTC_CNTR은 RPTC_LRC 값과 동일한 값에 도달한 후 증가하지 않습니다. 해제되면 RPTC_CNTR이 RPTC_LCR 레지스터의 값에 도달한 후 다시 시작됩니다.
5	R/W	0	INTE: 설정되면 PTC는 RPTC_CNTR 값이 RPTC_LRC 또는 RPTC_HRC의 값과 같을 때 인터럽트를 실행합니다. 신호가 지워지면 인터럽트가 마스킹됩니다.
6	R/W	0	INT: 읽을때 이 비트는 보류중인 인터럽트를 나타냅니다. 설정되면 인터럽트가 보류됩니다. 이 비트를 '1'로 쓰면 인터럽트 요청이 해제됩니다.
7	R/W	0	CNTRRST: 설정되면 RPTC_CNTR이 재설정됩니다. 지워지면 카운터가 정상적으로 작동합니다.
8	R/W	0	CAPTE: 설정되면 RPTC_CNTR이 RPTC_LRC 또는 RPTC_HRC 레지스터로 캡처됩니다. 해제되면 캡처 기능이 마스킹됩니다.

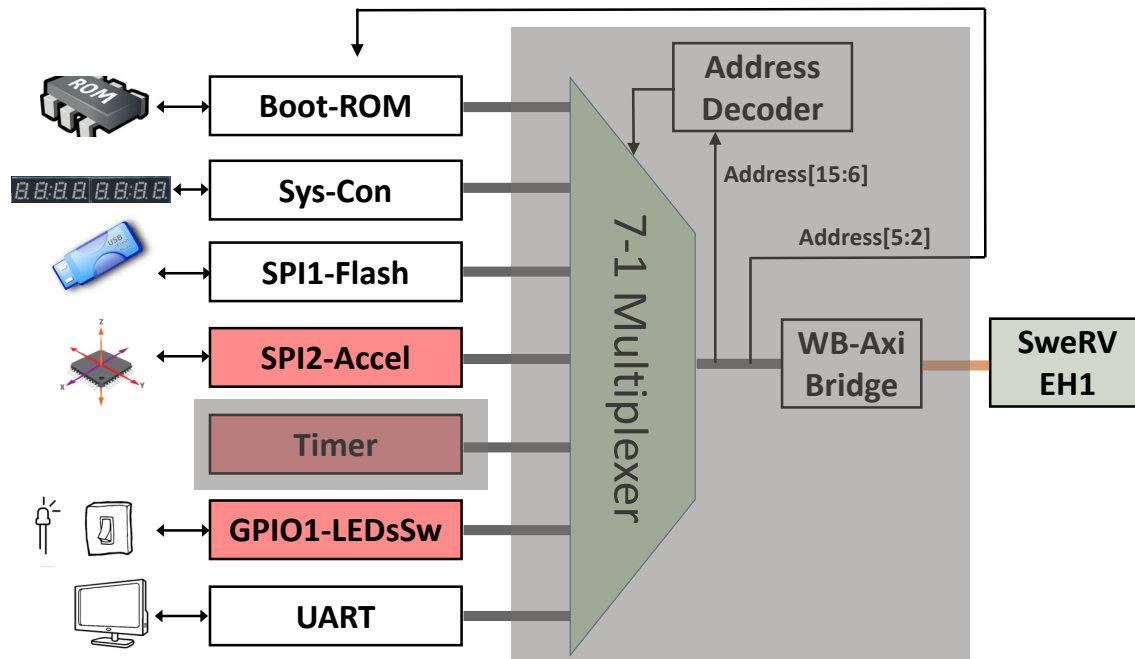
RVfpga Lab 8: 타이머 예제

- RPTC_LRC를 카운트할 사이클 수로 설정
- **타이머를 구성하려면 제어 비트 (RPTC_CTRL)를 설정하십시오.**
 - 카운터 재설정 및 인터럽트 지우기: RPTC_CTRL = 0xC0 (0b011000000): CNTRRST (비트 7) = 1: 카운터가 재설정 됨; (RPTC_CNTR = 0); INT (비트 6) = 1: 인터럽트 요청이 해제되었습니다.
 - 카운터 및 인터럽트 활성화: RPTC_CTRL = 0x21 (0b000100001): EN (비트 0) = 1: 카운터가 활성화되어 RPTC_CNTR이 증가합니다. INTE (비트 5) = 1: PTC는 RPTC_CNTR == RPTC_LRC 일 때 인터럽트를 실행합니다.
- 프로그램은 제어 레지스터 (INT는 RPTC_CTRL의 비트 6)에서 인터럽트 비트를 1 (RPTC_CNTR == RPTC_LRC를 나타냄)이 될 때까지 읽습니다.
- 이 알고리즘은 인터럽트를 사용하지 않지만 인터럽트 비트 (INT, RPTC_CTRL의 비트 6)를 읽어 정확한 클럭 사이클 수에 도달한 시기를 결정합니다. Lab 9에서 인터럽트를 사용하는 방법을 보여줍니다.



RVfpga Lab 8: 타이머 초급 수준 구현

- 2 개의 주요 부분으로 나뉩니다.
 - (외부 연결 없음)
 - 타이머 모듈을 SweRVofX에 통합 (왼쪽 음영 영역)
 - 타이머와 SweRV EH1 간의 연결 (오른쪽 음영 영역)



RVfpga Lab 8: SweRVolfX에 통합

파일 swervolf_core.v: PTC 모듈 인스턴스화

```
// PTC
wire      ptc_irq;

ptc_top timer_ptc(
    .wb_clk_i      (clk),
    .wb_rst_i      (wb_rst),
    .wb_cyc_i      (wb_m2s_ptc_cyc),
    .wb_adr_i      ({2'b0,wb_m2s_ptc_adr[5:2],2'b0}),
    .wb_dat_i      (wb_m2s_ptc_dat),
    .wb_sel_i      (4'b1111),
    .wb_we_i      (wb_m2s_ptc_we),
    .wb_stb_i      (wb_m2s_ptc_stb),
    .wb_dat_o      (wb_s2m_ptc_dat),
    .wb_ack_o      (wb_s2m_ptc_ack),
    .wb_err_o      (wb_s2m_ptc_err),
    .wb_inta_o     (ptc_irq),
    // External PTC Interface
    .gate_clk_pad_i (),
    .capt_pad_i  (),
    .pwm_pad_o  (),
    .oen_padoen_o ()
);
```



Lab 9:

인터럽트 구동 I/O



RVfpga Lab 9: 인터럽트 구동 I/O

- 인터럽트 구동 I/O vs. 프로그래밍 된 I/O
- Rvfpga 시스템의 인터럽트 컨트롤러
- Western Digital의 주변기기 지원 및 보드 지원 패키지 (PSP 및 BSP)를 사용하여 인터럽트를 구성하는 방법
- 인터럽트 예제

RVfpga Lab 9: 인터럽트 기반 I/O 소개

- **프로그래밍된 I/O:**

- 프로그램은 원하는 값이 표시 될 때까지 값 (예: 스위치)을 지속적으로 폴링 합니다.
- 예를 들어,이 방법은 이전 LAB에서 스위치를 읽는 데 사용되었습니다.
- 이것은 다른 유용한 작업을 수행할 수 있는 대신 지속적으로 값을 폴링하여 프로세서를 묶습니다.

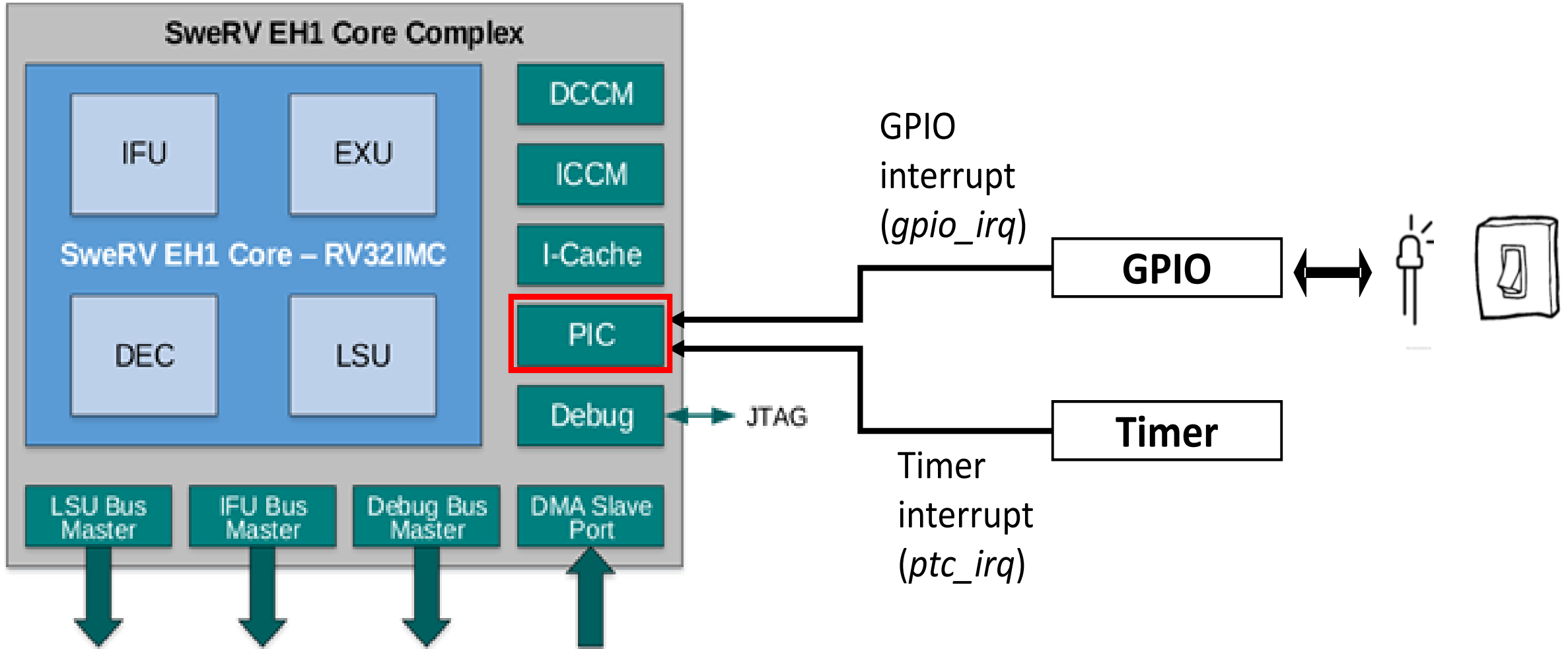
- **인터럽트 구동 I/O:**

- 이벤트 (예: 핀 설정)는 프로세서가 예약되지 않은 함수 호출과 유사한 인터럽트 서비스 루틴 (ISR, 인터럽트 핸들러라고도 함)으로 점프하게 합니다. ISR은 인터럽트를 처리합니다. 예를 들어 스위치 값을 읽은 다음 일반 프로그램으로 돌아갑니다.
- 해당 이벤트가 발생할 때까지 프로세서는 유용한 작업을 계속할 수 있습니다.

RVfpga Lab 9: 인터럽트 처리

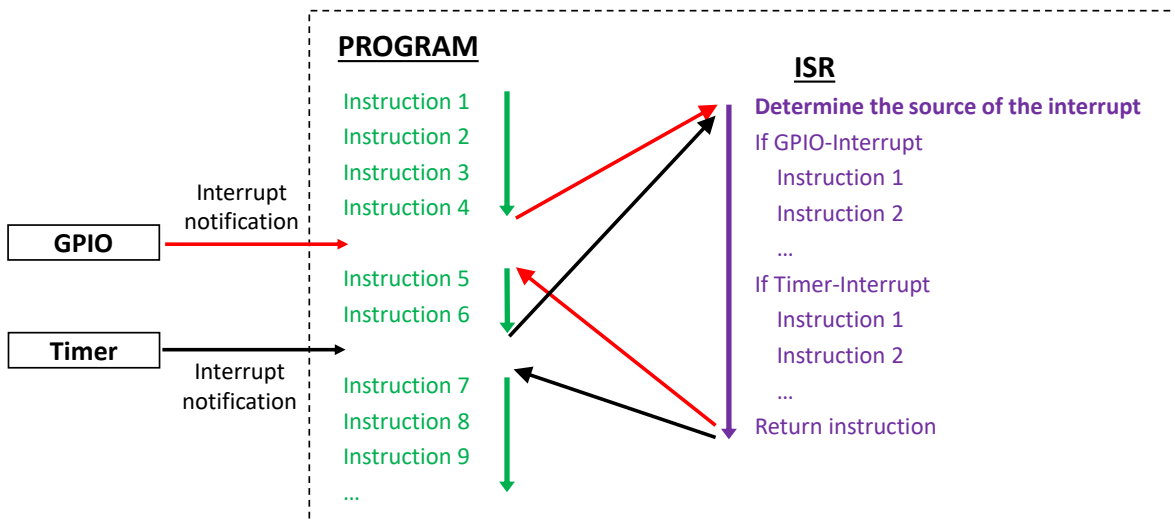
- 중단은 하드웨어 또는 소프트웨어로 인해 발생할 수 있습니다.
- 이 LAB에서는 하드웨어 인터럽트에 중점을 둡니다.
- SweRV EH1 코어는 RISC-V의 PLIC (플랫폼 수준 인터럽트 컨트롤러) 사양 이후에 인터럽트를 처리합니다. 이를 PIC (Programmable Interrupt Controller)라고 합니다.
 - 255 개의 인터럽트 소스
 - 15 가지 우선 순위 수준

RVfpga Lab 9: 인터럽트 하드웨어

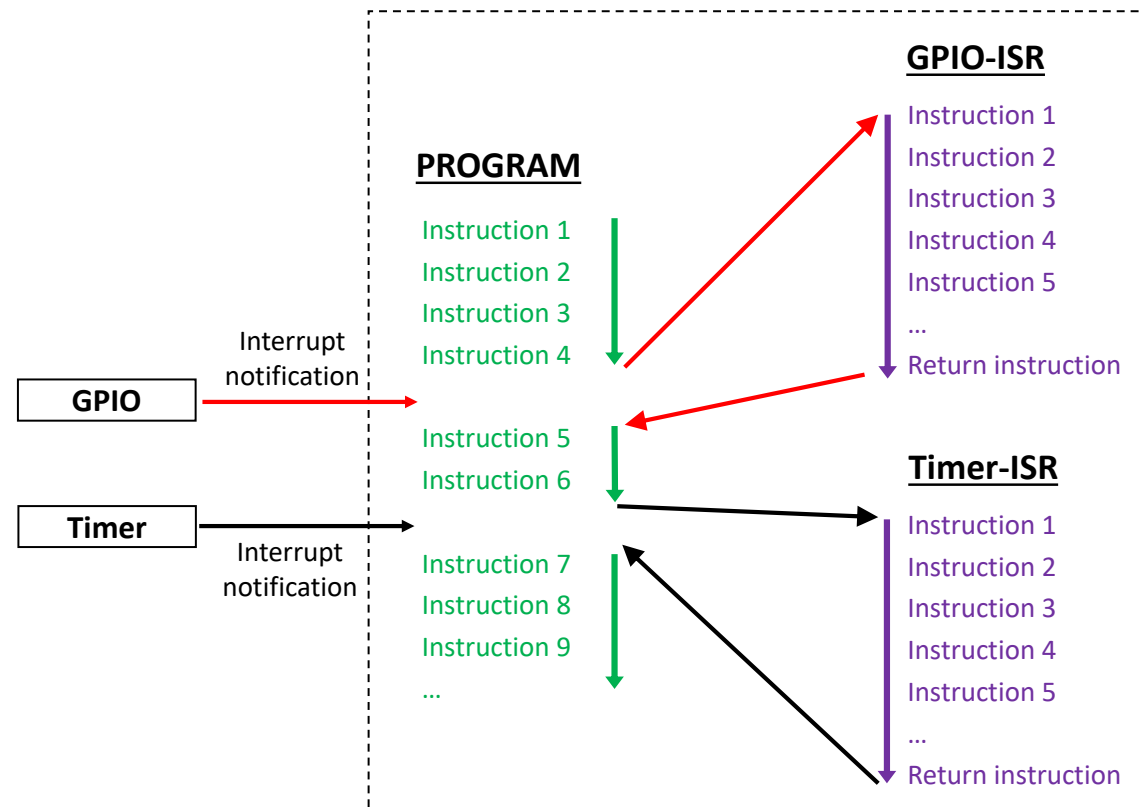


RVfpga Lab 9: 단일 벡터 vs. 다중 벡터 모드

단일 벡터 모드 예 :



다중 벡터 모드 예 :



RVfpga Lab 9: 인터럽트 처리

- WD's PSP/BSP 사용:
 - WD 'S PSP / BSP를 사용하여 인터럽트 초기화
 - 255 개 인터럽트 중 하나 이상을 초기화하고 ISR 이름 제공
 - 인터럽트를 트리거해야 하는 주변 신호를 인터럽트 핀으로 연결합니다.
 - 모든 인터럽트 활성화
 - 외부 인터럽트 활성화

RVfpga Lab 9: 인터럽트 예제

- 인터럽트를 사용하여 Switch [0]의 값 읽기 – 상승 에지에서만 (0→1 전환)

Name	Address	Width	Access	Description
RGPIO_IN	0x80001400	1-32	R	GPIO input data
RGPIO_OUT	0x80001404	1-32	R/W	GPIO output data
RGPIO_OE	0x80001408	1-32	R/W	GPIO output driver enable
RGPIO_INTE	0x8000140C	1-32	R/W	Interrupt enable
RGPIO_PTRIG	0x80001410	1-32	R/W	Type of event that triggers an interrupt
RGPIO_AUX	0x80001414	1-32	R/W	Multiplex auxiliary inputs to GPIO outputs
RGPIO_CTRL	0x80001418	2	R/W	Control register
RGPIO_INTS	0x8000141C	1-32	R/W	Interrupt status
RGPIO_ECLK	0x80001420	1-32	R/W	Enable gpio_eclk to latch RGPIO_IN
RGPIO_NEC	0x80001424	1-32	R/W	Select active edge of gpio_eclk

전체 코드는 다음을 참조하십시오:

[RVfpgaPath]/RVfpga/Labs/Lab9/LED-Switch_7SegDispl_Interrupts_C-Lang.c

RVfpga Lab 9: 인터럽트 예제

- 인터럽트 용 GPIO 레지스터 설정:
 - RGPIO_INTE = 0x10000 (스위치 [0]에 대한 인터럽트 활성화)
 - RGPIO_PTRIG = 0x10000 (Switch [0]의 상승 에지에서 인터럽트가 트리거 됨)
 - RGPIO_INTS = 0x0 (모든 인터럽트 지우기)
 - RGPIO_CTRL = 0x1 (GPIO 인터럽트 활성화)

RVfpga Lab 9: 인터럽트 예제

- GPIO ISR:

```
void GPIO_ISR(void) {
    unsigned int i;

    /* Invert LED value */
    i = M_PSP_READ_REGISTER_32(GPIO_LEDs);      /* RGPIO_OUT */
    i = !i;                                       /* Invert the LEDs */
    i = i & 0x1;                                 /* Only keep right-most LED */
    M_PSP_WRITE_REGISTER_32(GPIO_LEDs, i)        /* RGPIO_OUT */

    /* Clear GPIO interrupt */
    M_PSP_WRITE_REGISTER_32(RGPIO_INTS, 0x0);   /* RGPIO_INTS */

    /* Stop the generation of this interrupt (IRQ4) */
    bspClearExtInterrupt(4);
}
```

RVfpga Lab 9: 인터럽트 예제

- 인터럽트 4 (IRQ4)를 스위치의 인터럽트와 연결하고 인터럽트 서비스 루틴을 GPIO_ISR로 설정합니다.
- 메모리 매핑 레지스터 $0x80001018 = 0x1$: GPIO 인터럽트를 IRQ4에 연결합니다.
- Global 인터럽트 활성화

Lab 10:

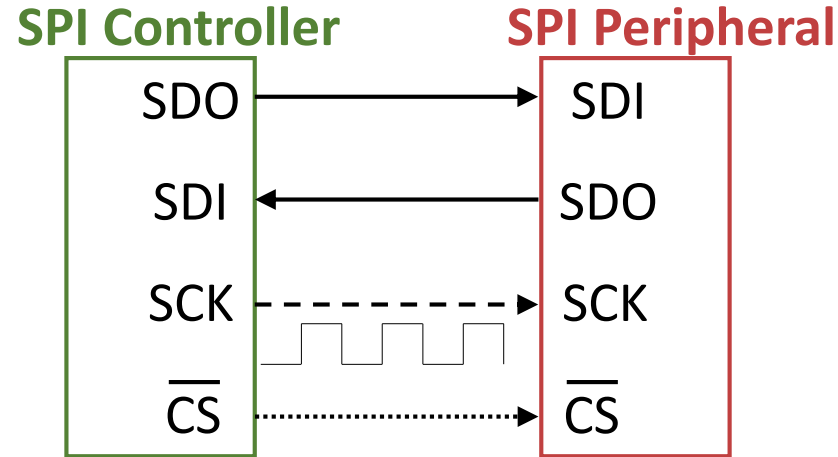
직렬 버스



RVfpga Lab 10: 직렬 버스

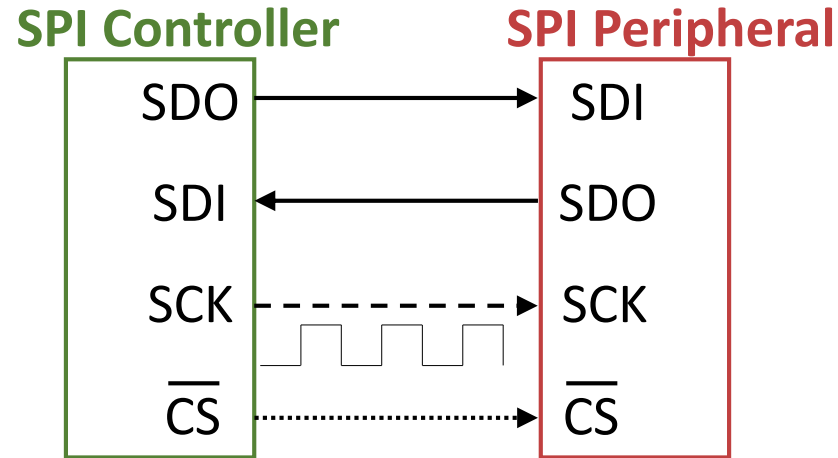
- 직렬 버스는 한 번에 1 비트 전송
 - 반대로 병렬 버스는 한 번에 여러 비트를 전송합니다.
 -
- 공통 직렬 버스
 - UART (범용 비동기 수신기 / 송신기)
 - SPI (직렬 주변기기 인터페이스)
 - I2C (통합 회로 프로토콜)
- 이 Lab에서는 **SPI**에 중점을 둡니다.

RVfpga Lab 10: 직렬 버스

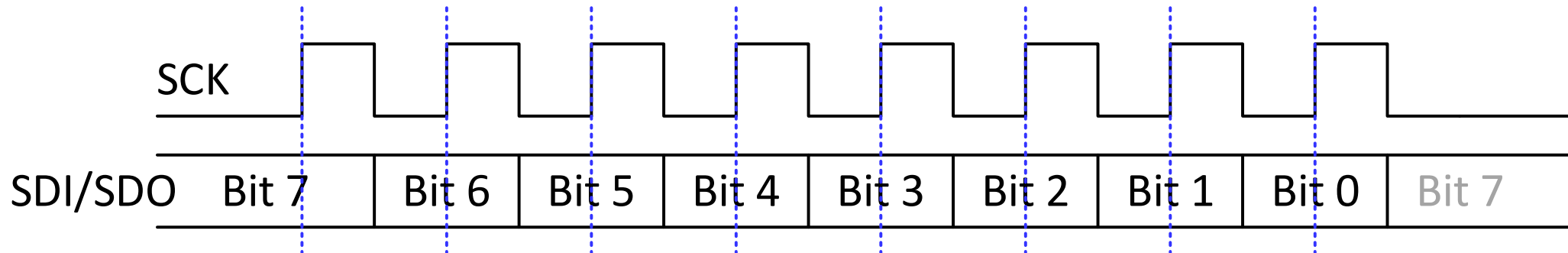


- 컨트롤러: 클럭을 보내고 데이터를 보내고 받습니다.
- 주변 장치: 클럭 수신, 데이터 송수신
- 신호:
 - SDO: 직렬 데이터 출력
 - SDI: 직렬 데이터 입력
 - SCK: SPI 클럭
 - CSbar: active low 칩 선택

RVfpga Lab 10: 직렬 버스



- **SCK 유희**
- 컨트롤러가 SCK에서 에지를 보내면 컨트롤러와 주변 장치 모두 샘플링하고 데이터를 보냅니다. 데이터는 하강 edge에서 변경 (전송)되고 상승 edge에서 샘플링 됩니다 (설정이 가능하지만).



RVfpga Lab 10: Rvfpga 시스템의 SPI 모듈

- Rvfpga 시스템의 SPI 모듈은 OpenCores에서 제공됩니다.

https://opencores.org/projects/simple_spi

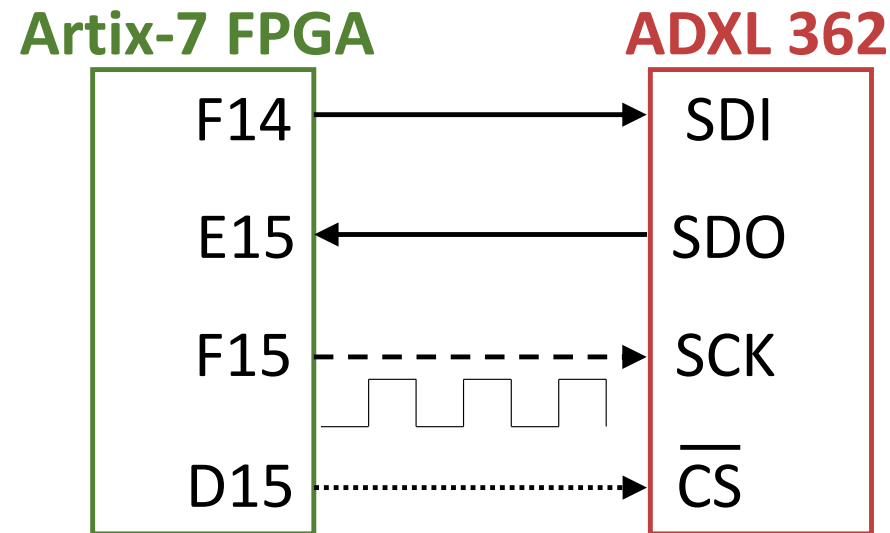
- 4 개 항목 읽기 및 쓰기 버퍼
- SPI 레지스터:

Name	Address	Width	Access	Description
SPCR	0x80001100	8	R/W	Control register
SPSR	0x80001108	8	R/W	Status register
SPDR	0x80001110	8	R/W	Data register
SPER	0x80001118	8	R/W	Extensions register
SPCS	0x80001120	8	R/W	CS register

RVfpga Lab 10: ADXL362 가속도계

- Nexys A7 보드에는 Analog Devices ADXL362 가속도계가 포함되어 있습니다. 전체 정보는 다음에서 찾을 수 있습니다.

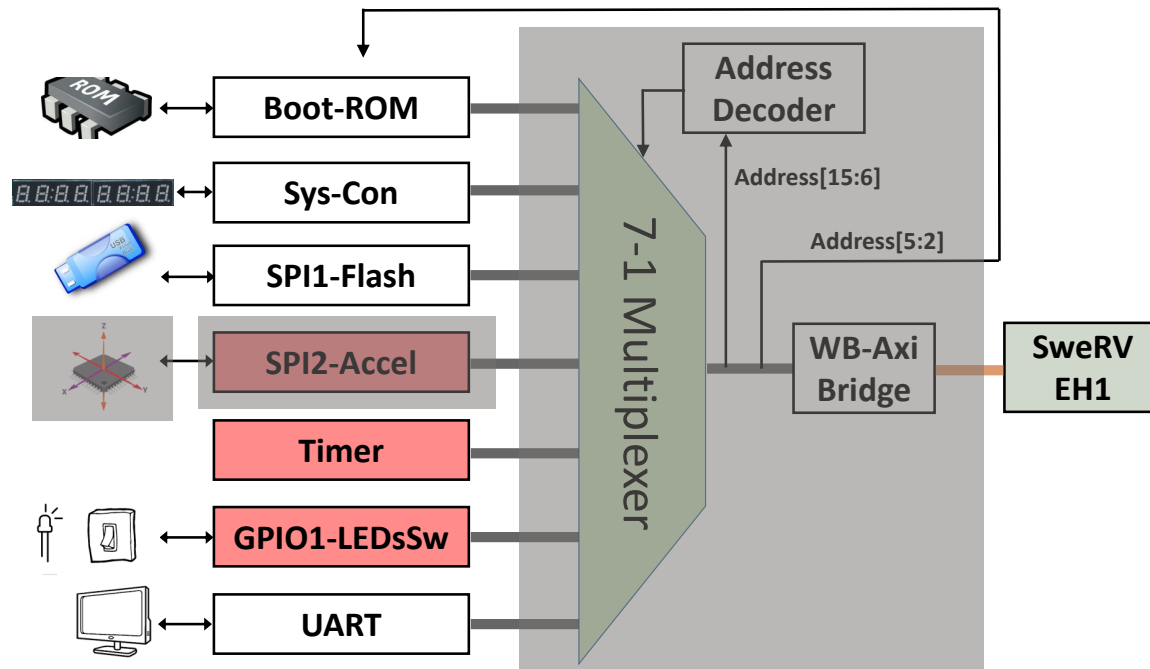
<https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>



RVfpga Lab 10: 가속도계. 초급 수준의 구현

- 3 개의 주요 부분으로 나뉩니다.

- 온보드 가속도계에 대한 RVfpgaNexys의 외부 연결 (왼쪽 음영 영역)
- 새로운 SPI 모듈을 SweRVolfX에 통합 (중간 음영 영역)
- 가속도계와 SweRV EH1 (오른쪽 음영 영역) 간의 연결



RVfpga Lab 10: 외부 연결

파일 rvfpganexys.xdc: SoC에서 사용되는 SPI 신호와 해당 온보드 가속도계 핀의 연결을 정의합니다.

```
78 ##Accelerometer
79 set_property -dict { PACKAGE_PIN E15   IOSTANDARD LVCMOS33 } [get_ports { i_accel_miso }]; #IO_L11P_T1_SRCC_15 Sch=acl_miso
80 set_property -dict { PACKAGE_PIN F14   IOSTANDARD LVCMOS33 } [get_ports { o_accel_mosi }]; #IO_L5N_T0_AD9N_15 Sch=acl_mosi
81 set_property -dict { PACKAGE_PIN F15   IOSTANDARD LVCMOS33 } [get_ports { accel_sclk }]; #IO_L14P_T2_SRCC_15 Sch=acl_sclk
82 set_property -dict { PACKAGE_PIN D15   IOSTANDARD LVCMOS33 } [get_ports { o_accel_cs_n }];
```

RVfpga Lab 10: SweRVolfX에 통합

파일 swervolf_core.v: Tri-state 버퍼 및 GPIO 모듈 인스턴스화

```
simple_spi spi2
// Wishbone slave interface
.clk_i    (clk),
.rst_i    (wb_rst),
.adr_i    (wb_m2s_spi_accel_adr[2] ? 3'd0 : wb_m2s_spi_accel_adr[5:3]),
.dat_i    (wb_m2s_spi_accel_dat[7:0]),
.we_i     (wb_m2s_spi_accel_we),
.cyc_i    (wb_m2s_spi_accel_cyc),
.stb_i    (wb_m2s_spi_accel_stb),
.dat_o    (spi2_rdt),
.ack_o    (wb_s2m_spi_accel_ack),
.inta_o   (spi2_irq),
// SPI interface
.sck_o    (o_accel_sclk),
.ss_o     (o_accel_cs_n),
.mosi_o   (o_accel_mosi),
.miso_i   (i_accel_miso));
```