



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 7

7 세그먼트 디스플레이

1. 소개

이 LAB에서는 RVfpga 시스템의 7 세그먼트 디스플레이에서 작동하도록 확장된 방법과 7 세그먼트 디스플레이 컨트롤러를 수정하는 방법을 보여줍니다. Nexys A7 FPGA 보드에는 8 자리 7 세그먼트 디스플레이가 있습니다. 먼저 작동 방식을 설명하고 (섹션 2) RVfpga 시스템 포함된 8 자리 7 세그먼트 디스플레이 컨트롤러의 고급 사양을 분석하고 몇 가지 기본적인 연습을 제공합니다 (섹션 3 및 4). 마지막으로 이 컨트롤러의 초급 수준의 구현을 분석하고 Verilator 시뮬레이션을 수행하며 컨트롤러 구현을 수정하고 실험할 추가 연습을 제공합니다 (섹션 5 및 6).

2. NEXYS A7 보드의 7 세그먼트 디스플레이

Nexys A7 보드에는 단일 8 자리 7 세그먼트 디스플레이로 작동하도록 구성된 2 개의 4 자리 공통 양극 7 세그먼트 LED 디스플레이¹가 포함되어 있습니다 (그림 1 참조). 8 자리는 각각 "그림 8" 패턴 (그림 2 참조)으로 배열된 7 개의 세그먼트로 구성되며 각 세그먼트에 대한 LED가 있습니다. 이러한 각 세그먼트는 켜거나 끌 수 있으므로 특정 LED 세그먼트를 비추고 나머지는 어둡게 두어 128 패턴 중 하나를 숫자에 표시할 수 있습니다. 특히 이 128 개의 패턴 중 10 진수는 그림 2와 같이 표시할 수 있습니다.

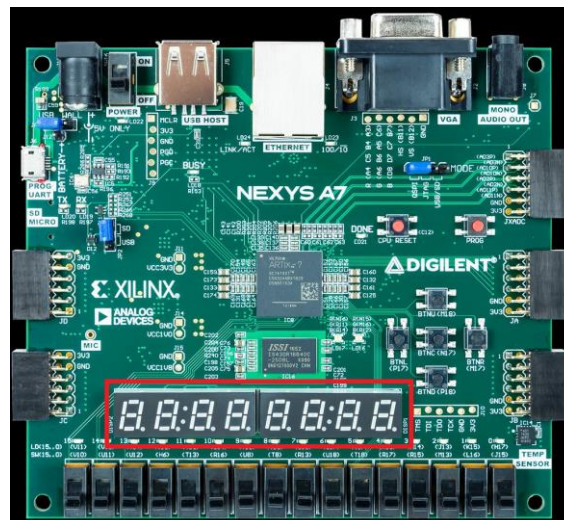


그림 1. Nexys A7의 8 자리 7 세그먼트 디스플레이



¹ The information in this section is described in: <https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual>

그림 2. 10 진수에 해당하는 패턴

한 자리 LED 세그먼트는 그림 3의 오른쪽에 표시된 것처럼 A-G로 표시되어 있습니다. 한 자리에 대한 7개의 LED의 양극은 하나의 "공통 양극" 회로 노드로 함께 연결되지만 LED 음극은 별도로 유지됩니다 (참조: 그림 3). 8개의 공통 양극 신호 (각 숫자 (AN0-AN7) 당 하나씩)는 "디지털 활성화" 역할을 합니다. 8자리 모두에서 동일한 세그먼트의 음극은 7개의 신호인 CA-CG로 (그림 3 참조) 연결됩니다. (소수점 DP에 대해 여덟 번째 신호가 존재하지만 이 LAB에서는 사용하지 않습니다.) 예를 들어, 8자리 숫자에서 세그먼트 D의 음극은 CD라는 단일 회로 노드로 함께 그룹화 됩니다. 이 신호 연결 방식은 음극 신호가 모든 숫자에 공통적인 다중 디스플레이를 생성하지만, 해당 양극 신호가 표시되는 숫자의 세그먼트만 조명할 수 있습니다. 이 모든 신호는 활성화되면 low로 구동됩니다. 따라서 세그먼트 (예: 숫자 2의 세그먼트 D)를 조명하려면 양극 AN2와 음극 CD를 모두 low로 구동해야 합니다.

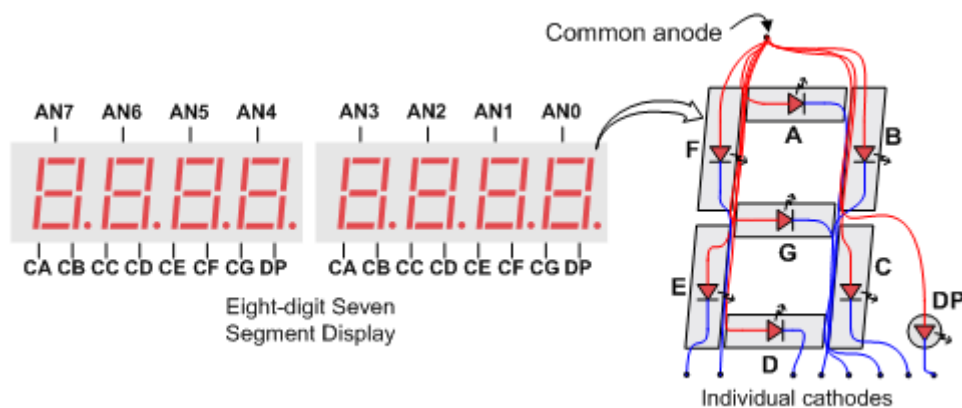


그림 3. Nexys A7의 8자리 7세그먼트 디스플레이 연결

디스플레이 컨트롤러 회로를 사용하여 7세그먼트 디스플레이에 8숫자를 표시할 수 있습니다. 이 회로는 사람의 눈으로 감지할 수 있는 것보다 더 빠른 업데이트 속도로 반복되는 연속 연속으로 각 숫자의 패턴으로 음극을 구동합니다. 동시에 회로는 양극을 한 번에 하나씩 구동합니다. 따라서 각 숫자는 8분의 1에 불과하지만 다시 밝아지기 전에는 숫자가 어두워지는 것을 눈으로 인식할 수 없기 때문에 숫자가 계속 켜져 있는 것처럼 보입니다.

8자리가 밝고 지속적으로 밝게 표시 되려면 8자리 모두 1 ~ 16ms 마다 한 번씩 구동해야 하며 각 자리는 새로 업데이트 주기의 1/8 동안 켜집니다 (예: 16ms 새로 업데이트 주기의 경우 숫자는 2ms 동안 켜집니다). 위에서 설명한 것처럼 컨트롤러는 해당 양극 신호도 low로 구동되는 동안 올바른 패턴으로 디지털 low의 음극을 구동해야 합니다. 그러나 Nexys A7은 NPN 트랜지스터를 사용하여 공통 양극 지점으로 충분한 전류를 구동하기 때문에 양극 활성화가 반전됩니다. 따라서 AN0~7 및 CA~G / DP 신호는 모두 활성 상태일 때 low로 구동됩니다.

프로세스를 설명하기 위해 맨 오른쪽 두 자리에 71을 표시한다고 가정합니다. 컨트롤러 회로는 처음 2ms 동안 AN0, CB 및 CC를 low로 구동하므로 맨 오른쪽 자리에 1이 표시됩니다. 이후 다음 2ms 동안 회로는 AN1, CA, CB 및 CC를 low로 구동하여 다음 최상위 숫자에 7을 표시합니다. 이 과정이 무한히 반복되면 사람의 눈은 맨 오른쪽 두 자리에 숫자 71이 표시됩니다.

3. 8-DIGIT 7-SEGMENT 디스플레이 컨트롤러의 높은 수준의 사양

이 섹션에서는 먼저 RVfpga 시스템에서 사용되는 8자리 7세그먼트 디스플레이 컨트롤러의 고급 사양을 설명하고 분석한 다음, 이를 사용하기 위한 연습을 제공합니다.

A. 고급 사양

이 과정에서 사용되는 7 세그먼트 디스플레이 컨트롤러는 RVfpga 시스템용으로 맞춤 설계 되었습니다. 여기에는 각각 주소 0x80001038 및 0x8000103C 에 매핑된 *Enables_Reg* 및 *Digits_Reg* 라는 두 개의 레지스터가 포함되어 있습니다 (이러한 주소는 시스템 컨트롤러 용으로 예약된 주소 범위 내에서 사용되지 않는 주소이며 <https://github.com/chipsalliance/Cores-SweRVolf/>에서 볼 수 있음).

작업: 레지스터 *Enables_Reg* 및 *Digits_Reg* 의 선언과 값이 할당된 위치를 찾습니다. 세그먼트 디스플레이는 파일로 구현됩니다. *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v*

Enables_Reg 는 각 비트가 해당 숫자가 *ON* (0)인지 *OFF* (1)인지 결정하는 8 비트 레지스터입니다. *Digits_Reg* 는 각 4 비트 그룹이 해당 숫자에 표시할 16 진수 값을 나타내는 32 비트 레지스터입니다. 예를 들어, 가장 오른쪽 두 자리에 71을 표시하려면 프로그래머는 레지스터에 다음 값을 할당합니다.

- *Enables_Reg* = 0xFC (맨 오른쪽 두 자리 활성화)
- *Digits_Reg* = 0x00000071 (값 = 71)

4. 기본 실습

연습 1. RISC-V 어셈블리 프로그램과 7 세그먼트 디스플레이의 맨 오른쪽 네 자리에 있는 스위치의 값을 보여주는 C 프로그램을 작성하십시오.

연습 2. RISC-V 어셈블리 프로그램과 7 세그먼트의 오른쪽에서 왼쪽으로 이동하는 "0-1-2-3-4-5-6-7-8" 문자열을 보여주는 C 프로그램을 작성합니다. 즉, 가장 오른쪽 자리에 0 이 먼저 표시되어야 합니다. 그런 다음 왼쪽으로 이동하고 맨 오른쪽 자리에 1 이 표시됩니다.

5. 8-DIGIT 7-SEGMENT 디스플레이 컨트롤러: 초급 수준의 구현, 시뮬레이션

지금까지 7 세그먼트 디스플레이 사용하는 방법을 살펴 보았습니다. 이 섹션에서는 간단한 어셈블리 예제를 실행할 때 초급 수준 구현을 설명하고 시뮬레이션에서 RVfpgaSim 를 분석합니다. 마지막으로 7 세그먼트 디스플레이 컨트롤러를 수정하는 연습을 제공합니다.

A. 7 세그먼트 디스플레이 컨트롤러의 초급 수준 구현

이전의 GPIO (범용 I/O) 실험실과 유사하게 7 세그먼트 디스플레이 컨트롤러의 분석을 세 단계로 나눕니다.

1. 보드의 SoC 와 I/O 장치 간의 연결 (그림 4 의 왼쪽 그림자 영역)
2. SoC 에 포함된 SweRVolfX 시스템 컨트롤러 내부에 포함된 새 컨트롤러의 통합 (그림 4 의 중간 그림자 영역)
3. 새 컨트롤러와 SweRV EH1 Core 간의 연결 (그림 4 의 오른쪽 그림자 영역).

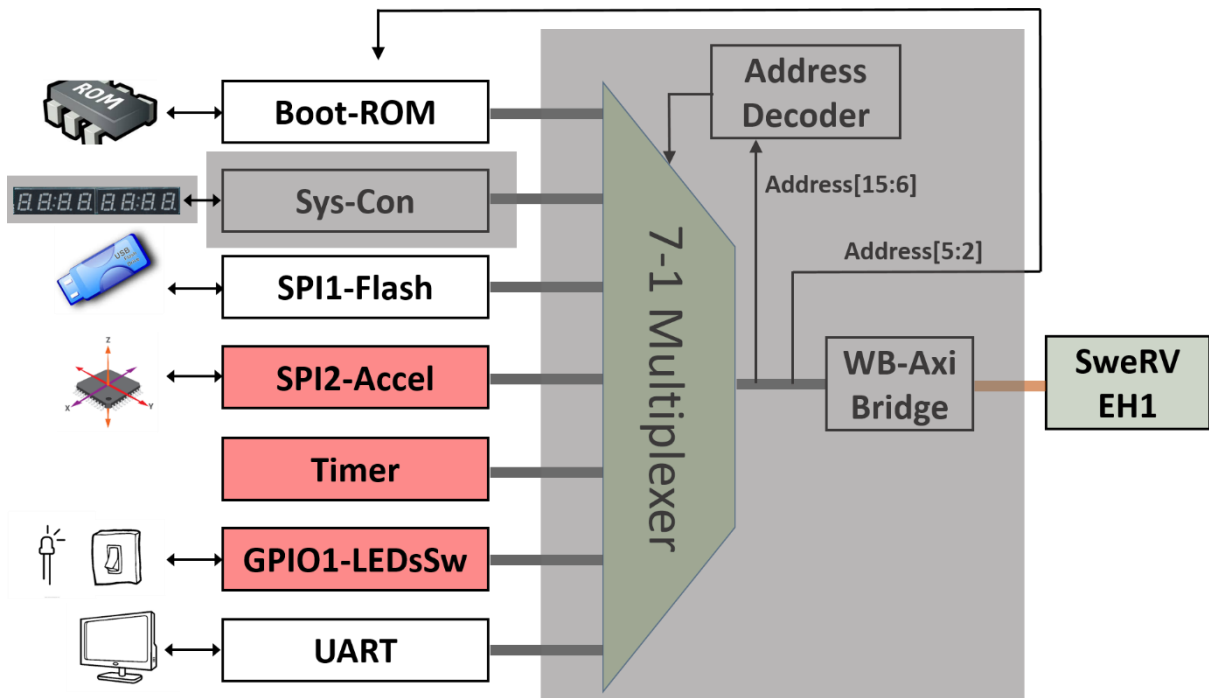


그림 4. 8 자리 7 세그먼트 디스플레이 컨트롤러 분석 3 단계

1. LED / 스위치를 SoC 에 연결

프로젝트의 constraints file 은 ([RVfpgaPath]/RVfpga/src/RVfpganexys.xdc) 입력/출력 SoC 신호와 보드 간의 연결을 정의 합니다. Nexys A7 FPGA 보드의 각 I/O 장치는 특정 FPGA 핀에 연결됩니다. 8 개의 양극 (그림 3 참조)을 연결하는 신호를 $AN[ij]$ (i 범위 0-7)라고 하고 8 자리 모두에서 유사한 세그먼트의 음극을 연결하는 신호 (그림 3 참조)를 CA 라고합니다. , CB , CC , CD , CE , CF 및 CG . 그림 5 는 이러한 연결이 정의된 제약 조건 파일의 일부분을 보여줍니다.

```
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]
```

그림 5. 8 자리 7 세그먼트의 연결은 입력을 표시합니다 (파일 RVfpganexys.xdc).

우리 시스템의 최상위 모듈 (모듈 RVfpganexys, [RVfpgaPath]/RVfpga/src/RVfpganexys.sv 파일에 구현 됨)의 50-51 행에서 SoC 에 연결된 7 세그먼트 디스플레이 입력 신호를 찾을 수 있습니다. $AN[ij]$ 및 $CA... CG$ (그림 6 의 왼쪽 부분), 해당 모듈의 끝 (그림 6 의 오른쪽 부분)에서 swervolf_core 모듈에 대한 연결을 찾을 수 있습니다 ($CA... CG$ 신호의 이름은 해당 모듈은 Digits_Bits [6:0])로 표시됩니다.

```

25 module rvfpganexys
26     #(parameter bootrom_file = "boot_main.mem")
27     (input wire      clk,
28      input wire      rstn,
29      output wire [12:0] ddram_a,
30      output wire [2:0] ddram_ba,
31      output wire      ddram_ras_n,
32      output wire      ddram_cas_n,
33      output wire      ddram_we_n,
34      output wire      ddram_cs_n,
35      output wire [1:0] ddram_dm,
36      inout wire [15:0] ddram_dq,
37      inout wire [1:0] ddram_dqs_p,
38      inout wire [1:0] ddram_dqs_n,
39      output wire      ddram_clk_p,
40      output wire      ddram_clk_n,
41      output wire      ddram_cke,
42      output wire      ddram_odt,
43      output wire      o_flash_cs_n,
44      output wire      o_flash_mosi,
45      input wire      i_flash_miso,
46      input wire      i_uart_rx,
47      output wire      o_uart_tx,
48      inout wire [15:0] i_sw,
49      output reg [15:0] o_led,
50      output reg [7:0] AN,
51      output reg      CA, CB, CC, CD, CE, CF, CG,

```

```

256     .i_ram_init_error (litedram_init_error),
257     .io_data           ((i_sw[15:0], gpio_out[15:0])),
258     .AN (AN),
259     .Digits_Bits ({CA, CB, CC, CD, CE, CF, CG}),
260     .o_accel_sclk      (accel_sclk),

```

그림 6. 8 자리 7 세그먼트 디스플레이를 SoC 에 연결 (파일: *rvfpganexys.sv*).

마지막으로 2 개의 신호가 **swervolf_core** 모듈에서 시스템 컨트롤러 모듈 (**swervolf_syscon**)로 삽입 됩니다 (그림 7 참조). 여기서 7 세그먼트 디스플레이 컨트롤러가 구현됩니다.

```

215 swervolf_syscon
216     #(.clk_freq_hz (clk_freq_hz))
217     syscon
218     (.i_clk           (clk),
219      .i_rst           (wb_rst),
220      .gpio_irq        (gpio_irq),
221      .ptc_irq         (ptc_irq),
222      .o_timer_irq     (timer_irq),
223      .o_sw_irq3       (sw_irq3),
224      .o_sw_irq4       (sw_irq4),
225      .i_ram_init_done (i_ram_init_done),
226      .i_ram_init_error (i_ram_init_error),
227      .o_nmi_vec       (nmi_vec),
228      .o_nmi_int       (nmi_int),
229      .i_wb_adr        (wb_m2s_sys_adr[5:0]),
230      .i_wb_dat        (wb_m2s_sys_dat),
231      .i_wb_sel        (wb_m2s_sys_sel),
232      .i_wb_we         (wb_m2s_sys_we),
233      .i_wb_cyc        (wb_m2s_sys_cyc),
234      .i_wb_stb        (wb_m2s_sys_stb),
235      .o_wb_rdt        (wb_s2m_sys_dat),
236      .o_wb_ack        (wb_s2m_sys_ack),
237      .AN (AN),
238      .Digits_Bits (Digits_Bits));
239

```

그림 7. 8 자리 7 세그먼트 디스플레이를 시스템 컨트롤러에 연결 (파일: *Swervolf_core.v*).

작업: 제약 조건 파일에서 시스템 컨트롤러 모듈 (CA-CG 가 *Digits_Bits* 배열로 병합 됨)으로 이 신호 (CA-CG 및 AN)를 따르십시오. 다음 파일을 확인하시기 바랍니다.

[RVfpgaPath]/RVfpga/src/RVfpganexys.xdc

[RVfpgaPath]/RVfpga/src/RVfpganexys.sv

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v

2. 8 자리 7 세그먼트 디스플레이 컨트롤러를 SoC 에 통합

모듈 **swervolf_syscon**

(*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v*)의 276-288 행에서 7 세그먼트 디스플레이 컨트롤러가 인스턴스화되고 SoC 에 통합됩니다 (그림 8 참조).

```

276 // Eight-Digit 7 Segment Displays
277
278 reg [ 7:0] Enables_Reg;
279 reg [31:0] Digits_Reg;
280
281 SevSegDisplays_Controller SegDispl_Ctr(
282     .clk          (i_clk),
283     .rst_n        (i_rst),
284     .Enables_Reg  (Enables_Reg),
285     .Digits_Reg   (Digits_Reg),
286     .AN           (AN),
287     .Digits_Bits  (Digits_Bits)
288 );
289
290 endmodule

```

그림 8. 8 자리 7 세그먼트는 컨트롤러 인스턴스화를 표시합니다 (파일: *swervolf_syscon.v*).

SevSegdisplays_Controller 모듈은 클럭 신호 (*i_clk*, *clk*로 이름 변경) 및 리셋 신호 (*i_rst*, *rst_n*으로 이름 변경) 외에도 두 개의 입력 신호 (*Enables_Reg* 및 *Digits_Reg*)를 수신합니다. 이 신호는 이미 설명한 두 개의 메모리 매핑 제어 레지스터입니다. 이 모듈은 보드의 7 세그먼트 디스플레이에 연결된 두 개의 신호 *AN* 및 *Digits_Bits*를 출력합니다. 맨 오른쪽 두 자리에 71 을 표시하는 예의 경우 **SevSegdisplays_Controller** 는 신호 *AN* 및 *Digits_Bits*에 다음 값을 할당합니다.

- 0 ~ 2ms: 신호 *AN[0]*이 low, 숫자 0 (가장 오른쪽 숫자)이 표시 되도록 합니다. 신호 *Digits_Bits[5]* 및 *Digits_Bits[4]* (*CB* 및 *CC*에 해당) 또한 숫자 0 (가장 오른쪽 숫자)에 "1"을 표시하도록 low 입니다. 다른 모든 신호는 high 입니다.
- 2 ~ 4ms: 신호 *AN[1]*이 low, 숫자 1 을 표시 할 수 있습니다. *Digits_Bits[6]*, *Digits_Bits[5]* 및 *Digits_Bits[4]* (*CA*, *CB* 및 *CC*에 해당)는 숫자 1 에 "7"을 표시하기 위해 high 입니다. 다른 모든 신호는 high 입니다.
- 4 ~ 16ms: *AN[2]... AN[7]*은 2ms 간격으로 높으므로 값을 표시하지 않습니다. 세그먼트는 나머지 자릿수인 2-7 자릿수에 대해서도 high 입니다.

SevSegdisplays_Controller 모듈은

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v 파일의 295-366 행에서 구현 됩니다. 여기에는 다음과 같은 서브유닛이 포함됩니다.

- 2 개의 멀티플렉서가 2ms 마다 *AN* 및 *Digits_Bits* 신호로 보낼 값을 선택합니다. 멀티플렉서는 **SevSegMux** 모듈 내부에서 구현됩니다.
- 2ms 주기를 생성하기 위해 우리는 *[RVfpgaPath]/RVfpga/src/OtherSources/pulp-platform.org_common_cells_1.20.0/src* 폴더에 포함된 *counter.sv* 및 *delta_counter.sv* 파일에 제공된 카운터 모듈을 사용합니다. 카운터는 0 에서 2^{19} 까지 카운트하도록 구성되며, 대략 2ms 마다 변경되는 3 개의 최상위 비트가 위에서 설명한 두 멀티플렉서의 선택 신호로 사용됩니다.
- 디코더는 주어진 4 비트 16 진수 값에 대한 세그먼트 값을 출력하는 모듈 **SevenSegDecoder** 에서 구현됩니다.

작업: **SevSegdisplays_Controller** 모듈을 자세히 분석합니다. 다음 섹션에서 수행되는 시뮬레이션은 이 작업에 도움이 될 수 있습니다. 필요한 경우 새 신호로 시뮬레이션을 확장할 수도 있습니다.

3. 8 자리 7 세그먼트 디스플레이 컨트롤러와 SweRV EH1 Core 연결

LAB 6 에 설명된 대로 장치 컨트롤러는 멀티플렉서를 사용하여 SweRV EH1 Core 에 연결됩니다 (그림 4 참조).

7:1 멀티플렉서 (그림 9)는 `[RVfpgaPath]/RVfpga/src` 파일의 104-205

행에서 (`/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh`) 인스턴스화되는

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v` 파일에서 구현됩니다. 이후 파일은 `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v` 위치에 있는 **swervolf_core** 모듈의 168 행에 포함되어 있습니다.

멀티플렉서는 주소에 (110~111) 따라 CPU (`wb_io_*` 신호 – 그림 9 의 115-126 행)를 Wishbone 버스 (그림 9 의 127-138 행)에 연결하여 읽거나 쓸 주변 장치를 선택합니다. 예를 들어 CPU 에서 생성된 주소가 0x80001000-0x8000103F 범위에 있으면 시스템 컨트롤러가 선택되므로 `wb_io_*` 신호는 `wb_sys_*` 신호와 연결됩니다.

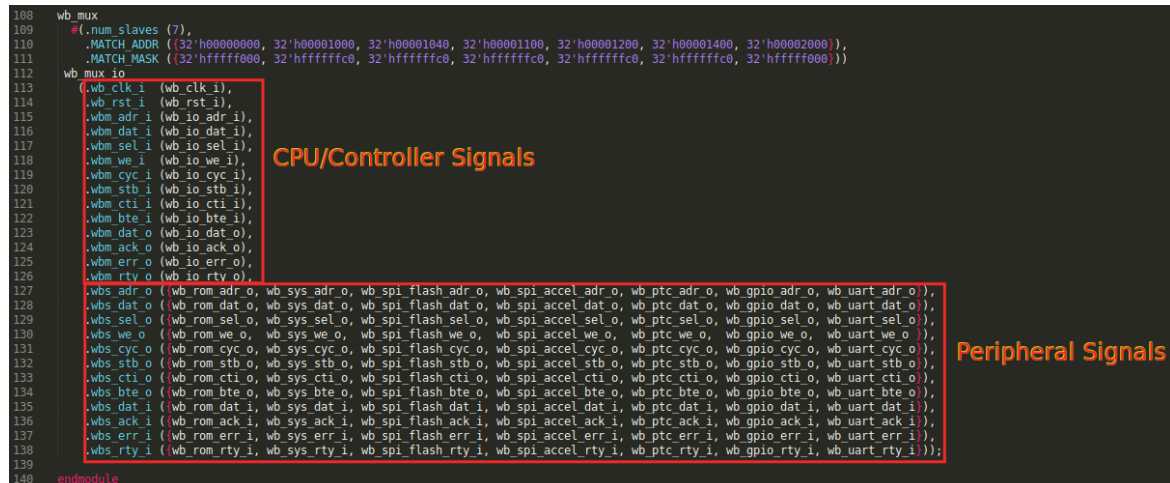


그림 9. 7-1 CPU 와 연결된 주변 장치를 선택하는 멀티플렉서 (파일: `wb_intercon.v`).

시스템 컨트롤러에 포함된 레지스터는 CPU 에서 생성된 주소 (`i_wb_adr`) (`swervolf_syscon` 모듈의 162-228 라인)를 기반으로 Wishbone 버스 (`i_wb_dat`)의 데이터 신호에 직접 연결하여 CPU 에서 기록됩니다.

작업: 시스템 컨트롤러에서 주소가 매핑되는 방식을 이해하기 위해 모듈 **swervolf_syscon** 의 162-228 행을 검사합니다. 레지스터 `Enables_Reg` 및 `Digits_Reg` 를 참조하는 219 ~ 227 행 (그림 10)에 중점을 둡니다 (앞에서 언급했듯이 이 두 레지스터에 할당된 주소는 각각 0x80001038 및 0x8000103C 임).

```

219      14 : begin
220          if (i_wb_sel[0]) Enables_Reg[7:0]  <= i_wb_dat[7:0];
221      end
222      15 : begin
223          if (i_wb_sel[0]) Digits_Reg[7:0]    <= i_wb_dat[7:0];
224          if (i_wb_sel[1]) Digits_Reg[15:8]   <= i_wb_dat[15:8];
225          if (i_wb_sel[2]) Digits_Reg[23:16]  <= i_wb_dat[23:16];
226          if (i_wb_sel[3]) Digits_Reg[31:24]  <= i_wb_dat[31:24];
227      end

```

그림 10. 8 자리 7 세그먼트 디스플레이와 Core 간의 연결 (파일 `swervolf_syscon.v`).

B. Verilator 시뮬레이션

이 섹션에서는 프로세서가 이 주변 장치를 구동하는 간단한 예제를 실행할 때 8 자리 7 세그먼트 디스플레이 컨트롤러의 주요 신호를 검사하기 위해 **RVfpgaSim** 을 사용합니다. 시뮬레이션에서 우리는 신호 *AN* 및 *Digits_Bits*를 분석하고 그림 11 의 예를 실행하여 맨 오른쪽 두 자리에 71 을 씁니다. 이 프로그램은 [RVfpgaPath]/RVfpga/Labs/Lab7/71_7SegDispl 에서 찾을 수 있습니다. C 버전은 [RVfpgaPath]/RVfpga/Labs/Lab7/71_7SegDispl_C-Lang 에서도 찾을 수 있습니다.

```
#define SegEn_ADDR    0x80001038
#define SegDig_ADDR   0x8000103C

.globl main
main:

    li t1, SegEn_ADDR
    li t6, 0xFC
    sb x6, 0(t1)                # Enable the 7SegDisplays

    li t1, SegDig_ADDR
    li t6, 0x71
    sw t6, 0(t1)                # Write the 7SegDisplays

next: beq zero, zero, next

.end
```

그림 11. 71_7SegDispl.S 예제

그림 12 는 71_7SegDispl.elf 프로그램의 디스 어셈블리 버전을 보여줍니다. 이 프로그램은 PlatformIO 에서 컴파일한 후 다음에서 찾을 수 있습니다.

[RVfpgaPath]/RVfpga/Labs/Lab7/71_7SegDispl/.pio/build/swervolf_nexys/firmware.dis

```
00000090 <main>:
 90: 80001337      lui    t1,0x80001
 94: 03830313      addi   t1,t1,56 # 80001038
 98: 0fc00f93      li     t6,252
9c: 01f30023      sb     t6,0(t1)
a0: 80001337      lui    t1,0x80001
a4: 03c30313      addi   t1,t1,60 # 8000103c
a8: 07100f93      li     t6,113
ac: 01f32023      sw     t6,0(t1)

000000b0 <next>:
b0: 00000063      beqz   zero,b0 <next>
```

그림 12. 71_7SegDispl.S 예제의 디스 어셈블리 버전

시뮬레이션을 실행하려면 다음 단계를 따르십시오. (시뮬레이션을 실행하지 않으려면 7 단계로 바로 이동할 수 있습니다.)

1. 이 경우 시뮬레이션의 경우에만 COUNT_MAX

(*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v* 파일의 295 행 참조)를 20 에서 5 로 변경하여 클럭 주기를 줄여야 합니다. 그렇지 않으면 결과를 보는 데 너무 많은 시간이 걸립니다. COUNT_MAX 의 값을 수정한 후 다음 명령을 실행하여 RVfpgaSim 을 다시 컴파일하십시오 (GSG 에 설명되어 있음).

```
cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean
make
```


새 파일 *Vrvfpgasim* (RVfpgaSim 시뮬레이션 바이너리)은 *[RVfpgaPath]/RVfpga/verilatorSIM* 디렉토리 내에 생성되어야 합니다.

WINDOWS: Windows 를 사용하는 경우 Cygwin 터미널에서 이러한 명령을 실행해야 합니다 (자세한 지침은 시작 안내서의 섹션 6 및 부록 C 참조). C:Windows 폴더는 Cygwin (*/cygdrive/c*)에서 찾을 수 있습니다.

MacOS: 자세한 지침은 시작 안내서의 부록 D 를 참조하십시오.

2. 컴퓨터에서 VSCode/PlatformIO 를 엽니다.
3. 상단 표시 줄에서 파일-폴더 열기를 클릭하고 *[RVfpgaPath]/RVfpga/Labs/Lab7* 디렉터리로 이동합니다.
4. *71_7SegDispl* 디렉토리를 선택하고 (열지 말고 선택 만) 확인을 클릭하십시오. 예제는 PlatformIO 에서 열립니다.
5. *platformio.ini* 파일을 열고 위에서 생성한 RVfpgaSim 시뮬레이션 바이너리의 경로가 올바른지 확인합니다. GSG 에서 다음과 같이 표시되어야 합니다:

```
board_debug.verilator.binary =
[RVfpgaPath]/RVfpga/verilatorSIM/Vrvfpgasim
```

6. 왼쪽 메뉴 리본에서 PlatformIO 아이콘  을 클릭하여 시뮬레이션을 실행한 다음 Project Tasks → env: swervolf_nexys → Platform 을 확장하고 Generate Trace 를 클릭합니다.

trace.vcd 파일은 *[RVfpgaPath]/RVfpga/Labs/Lab7/71_7SegDispl/.pio/build/swervolf_nexys* 내에 생성되어야 하며 다음 명령을 실행하여 *GTKWave* 로 열 수 있습니다.

```
gtkwave [RVfpgaPath]/RVfpga/Labs/Lab7/71_7SegDispl/.pio/build/swervolf_nexys/trace.vcd
```

WINDOWS: 다운로드한 *gtkwave64* 폴더의 bin 폴더 안에 *gtkwave.exe* 라는 응용 프로그램이 포함되어 있습니다. 해당 응용 프로그램을 두 번 클릭하여 GTKWave 를 시작합니다. 응용 프로그램의 상단에서 **파일 – 새 탭 열기**를 클릭하고 *[RVfpgaPath]/RVfpga/Labs/Lab7/71_7SegDispl/.pio/build/swervolf_nexys* 폴더에 생성된 *trace.vcd* 파일을 엽니다.

7. 시뮬레이션에 다음 신호를 포함합니다 (각 신호를 찾기 위해 참조된 모듈로 이동).
 - rvfpgasim – swervolf – syscon – SegDispl_Ctr
 - ✓ 입력 신호: **Enables_Reg** and **Digits_Reg**.
 - ✓ 출력 신호: **AN** and **Digits_Bits**.
8. 그림 13 에 표시된 시뮬레이션을 분석합니다. 처음에 8 개의 7 세그먼트 디스플레이에 표시된 값은 모두 0 입니다 (초기에는 모든 숫자가 *Enables_Reg*=0 으로 활성화 됨). 그런 다음 *Enables_Reg* (그림 12 의 sb

명령어)에 `0xFC`를 작성하여 맨 왼쪽 6 자리 숫자를 비활성화하고 `Digits_Reg` (그림 12의 `sw` 명령어)에 `0x71`을 작성하여 맨 오른쪽 2 자리 숫자에 `71`을 작성합니다. 출력 신호에 미치는 영향은 다음과 같습니다 (그림 13 참조).

- 첫 번째 기간: `AN=0xFE` 및 `Digits_Bits=0x4F`, 따라서 맨 오른쪽 숫자인 숫자 0에 1이 표시됩니다.
- 두 번째 기간: `AN=0xFD` 및 `Digits_Bits=0x0F`, 따라서 다음 숫자인 숫자 1에 7이 표시됩니다.
- 다음 6개 기간: `AN=0xFF` 및 `Digits_Bits=0x01`, 따라서 가장 왼쪽에 있는 6자리 숫자를 끄니다.
- 이 과정이 반복됩니다.

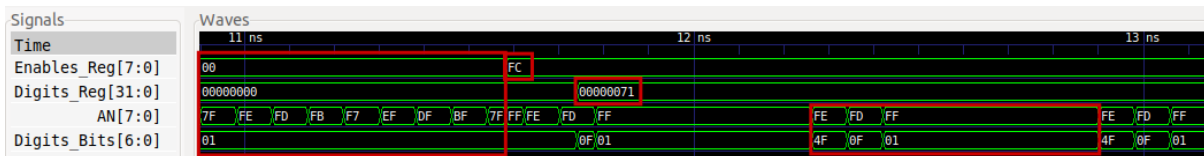


그림 13. 8 자리 7 세그먼트 디스플레이의 가장 오른쪽 두 자리에 값 71 쓰기

9. 계속하기 전에 `COUNT_MAX`의 값을 원래 값 (`COUNT_MAX = 20`)으로 복원하는 것을 잊지 마십시오.

6. 고급 실습

연습 3. 7 세그먼트 디스플레이에 ON/OFF LED 조합을 표시할 수 있도록 이번 LAB에서 설명한 컨트롤러를 수정합니다.

- 지금 활성화 레지스터가 필요하지 않습니다. 대신 8개의 7비트 레지스터가 필요합니다. 8자리 7 세그먼트 디스플레이 각각의 세그먼트를 `Segments_Digit0` ~ `Segments_Digit7`라 부릅니다. 각 레지스터에서 각 비트는 해당 세그먼트가 ON (0)인지 OFF (1)인지를 나타냅니다. 예를 들어, 첫 번째 레지스터 (`Segments_Digit0`)의 모든 비트가 0이면 맨 오른쪽 숫자의 모든 세그먼트가 ON이 되고 첫 번째 레지스터의 모든 비트가 1이면 맨 오른쪽 숫자의 모든 세그먼트가 OFF됩니다.
- 이 두 개의 새 레지스터를 이전에 사용한 것과 동일한 주소에 매핑할 수 있습니다 (먼저 이전 레지스터 `Enables_Reg` 및 `Digits_Reg` 두 개를 제거).
 - `Segments_Digit0` ↔ Address `0x80001038`
 - `Segments_Digit1` ↔ Address `0x80001039`
 - ...
 - `Segments_Digit7` ↔ Address `0x8000103F`
- 프로그램에서 제공하는 정보가 이미 디코딩 되었으므로 더 이상 4-7 디코더 (모듈 **SevenSegDecoder**)가 필요하지 않습니다.

연습 4. 새 컨트롤러를 사용하여 8자리 7 세그먼트 디스플레이에 "I SAY HI"를 인쇄합니다. 평소와 같이 프로그램의 RISC-V 어셈블리와 C 버전을 모두 구현합니다.