SUSTech Southern University of Science and Technology

SME-309
Microprocessor Design

Lab 1 – Familiarization with Keil MDK and Vivado

Content

SUSTech Southern University of Science and Technology

1. Background and introduction
2. Keil Quick Start
3. Debug with Keil
4. Seven-Segment Display
5. Vivado Tips
6. Lab1 Task

Content

SUSTech Southern University of Science and Technology

1. **Background and introduction**
2. Keil Quick Start
3. Debug with Keil
4. Seven-Segment Display
5. Vivado Tips
6. Lab1 Task

Background

Keil MDK

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

Assembly language program (for ARMv8)

```
swap:
        LSL  X10, X1,3
        ADD  X10, X0,X10
        LDUR X9, [X10,0]
        LDUR X11,[X10,8]
        STUR X11,[X10,0]
        STUR X9, [X10,8]
        BR   X10
```

Assembler

Binary machine language program (for ARMv8)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

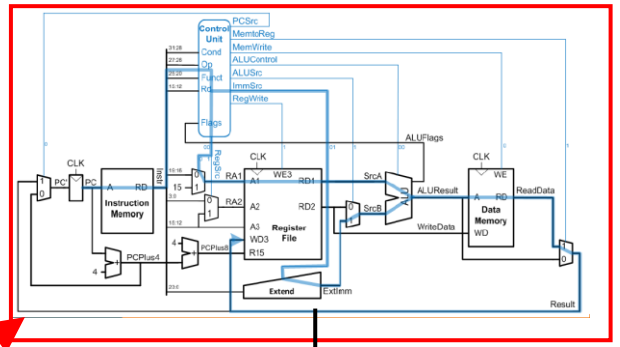
Integrated development environments
…Include C/C++ compilers
…For Arm Cortex®-M based devices.

Assembly Code (.s)
Very low-level, still readable
Based on CPU Instruction Set

Provided Instructions for CPU to Run
Trick: Hard coding into Verilog

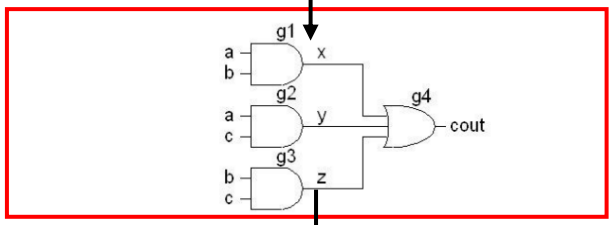
Machine Code (.hex)
For CPU to read & run

Vivado



Verilog Code (.v)
Hardware Description
- eg, simple CPU

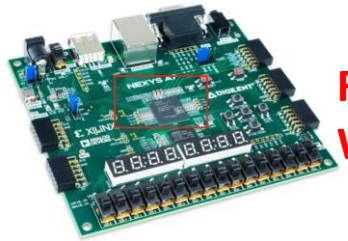
synthesize



Netlist

Implement

Generate bit files, Mapping to FPGA



Run *.c/*.s code in FPGA,
Which is CPU designed by *.v

Introduction

In this lab, you will get familiar with Keil MDK and Vivado.

Keil MDK (Microcontroller Development Kit) is the complete software development environment for a range of Arm Cortex-M based microcontroller devices. MDK includes the µVision IDE and debugger, Arm C/C++ compiler, and essential middleware components.

You can use Keil MDK to do assembly language programming, simulation, and even compilation of C++.

In this lab, we will learn to use Keil MDK to **build a project**, **debug**, and **generate HEX file**.

Then the HEX file can be converted to Verilog file, which will be used to initialize the ROM of your CPU (by Hex2ROM.exe). It can also be used to check whether your ARM CPU can run proper instructions by "initializing" your instruction memory file when running the simulation in later labs.

Files to be needed:

1. MDK-523.exe
2. MDK79523.exe
3. Sample_ASM_Keil_ARM7.s
4. MMIO.ini
5. Hex2ROM.exe

Content

SUSTech Southern University of Science and Technology

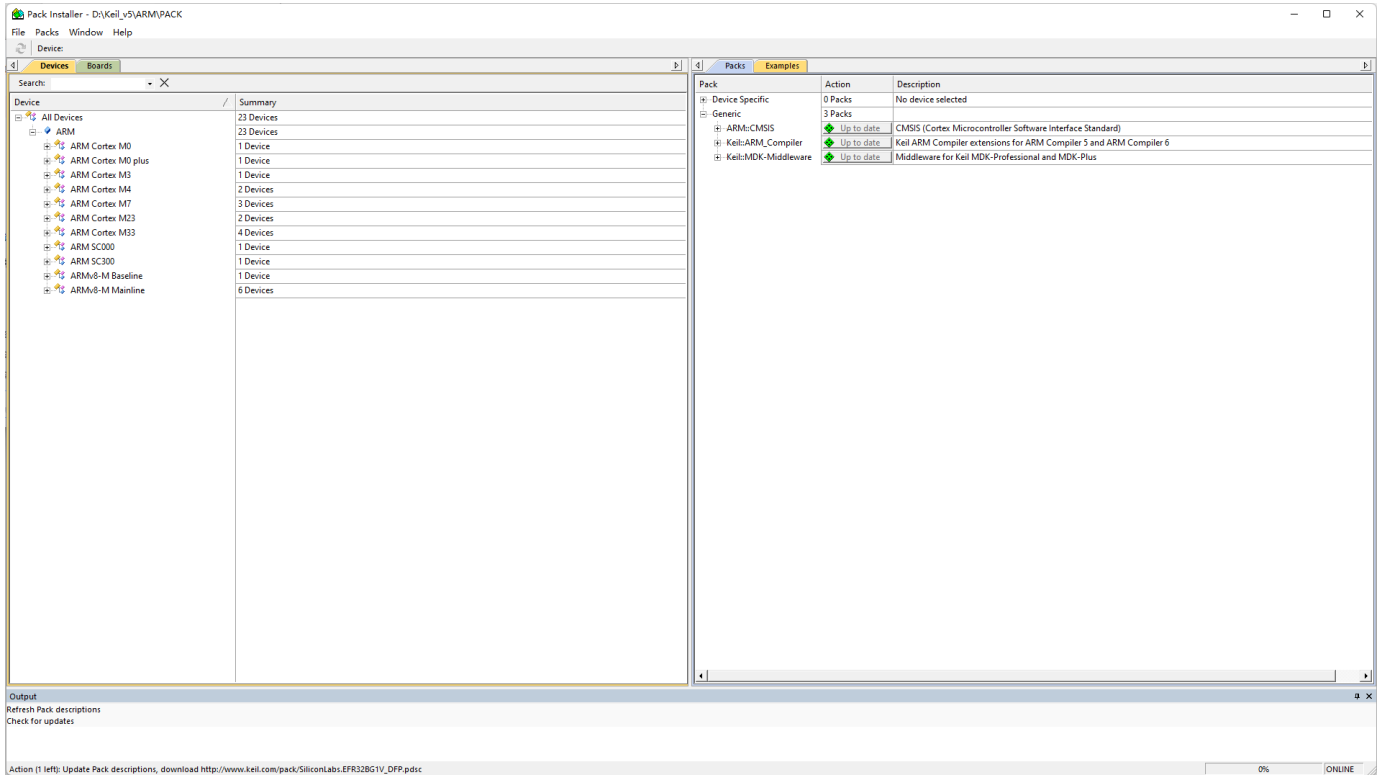
1. Background and introduction
2. **Keil Quick Start**
3. Debug with Keil
4. Seven-Segment Display
5. Vivado Tips
6. Lab1 Task

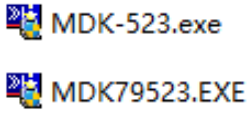
Keil Quick Start

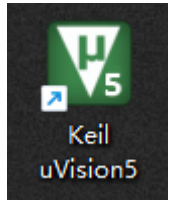
1. Install Keil.

   Step1: Install MDK-523.exe;

   Step2: Install MDK79523.exe (ARMv3 patch);

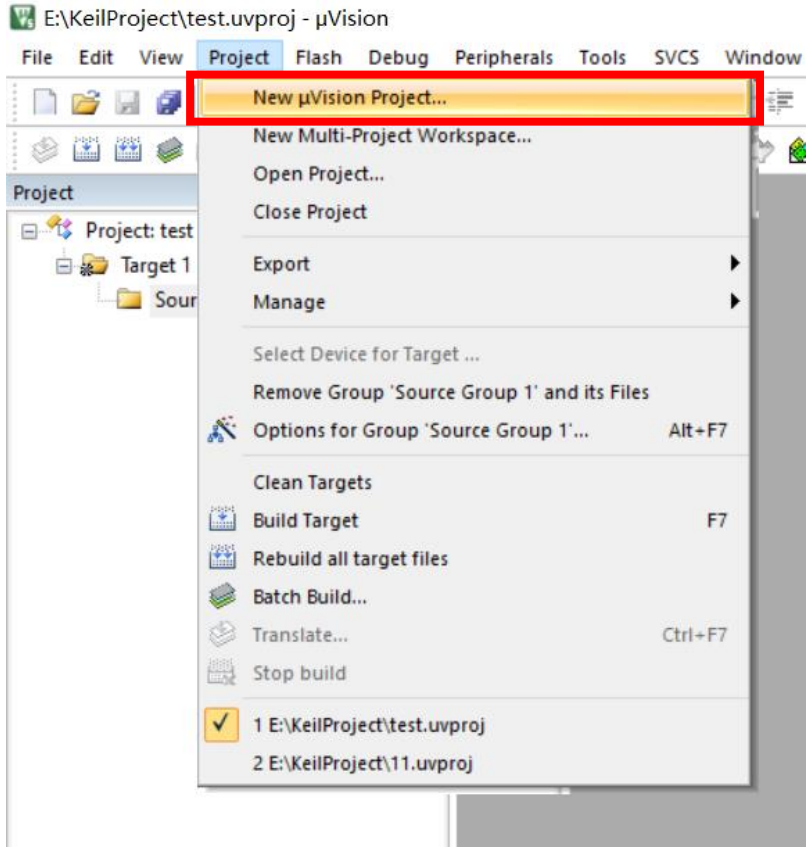


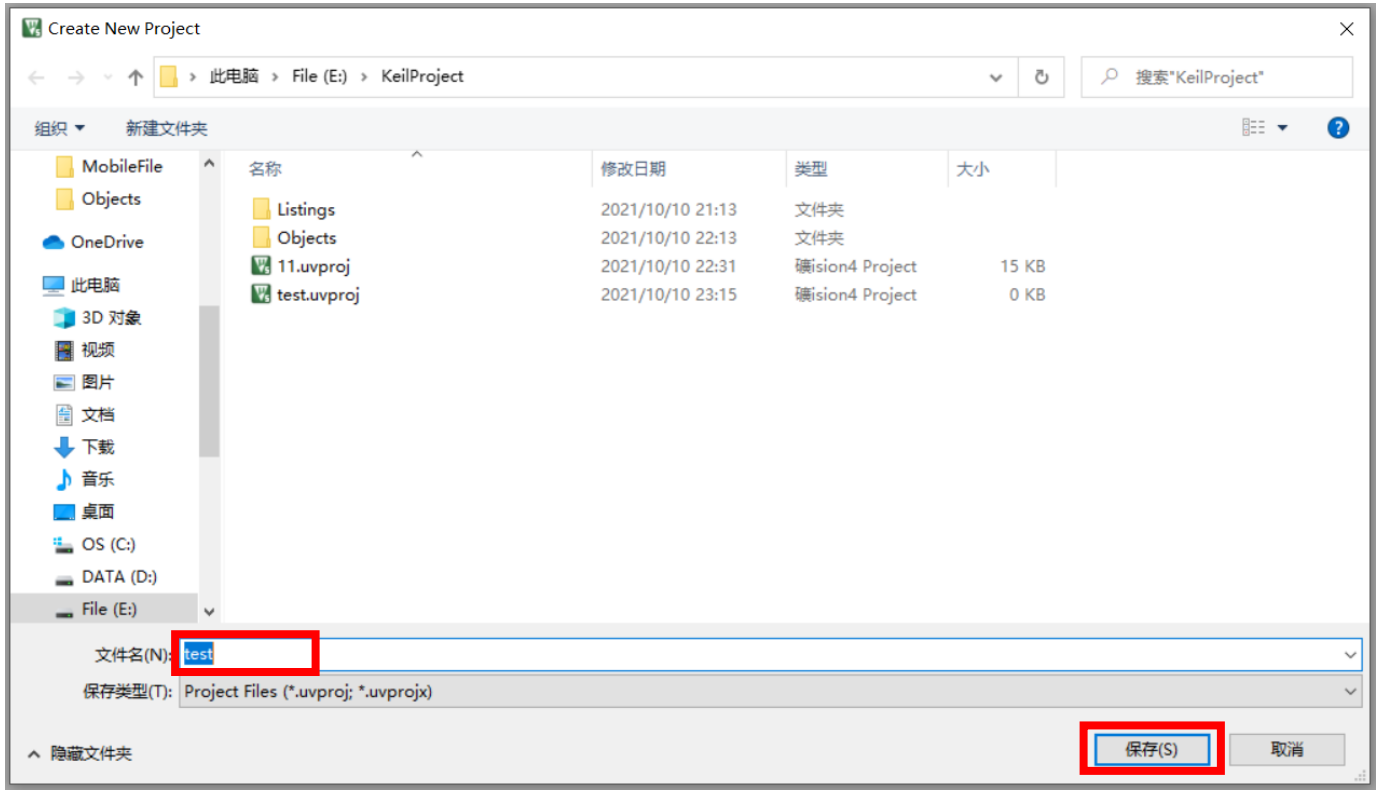




Keil Quick Start

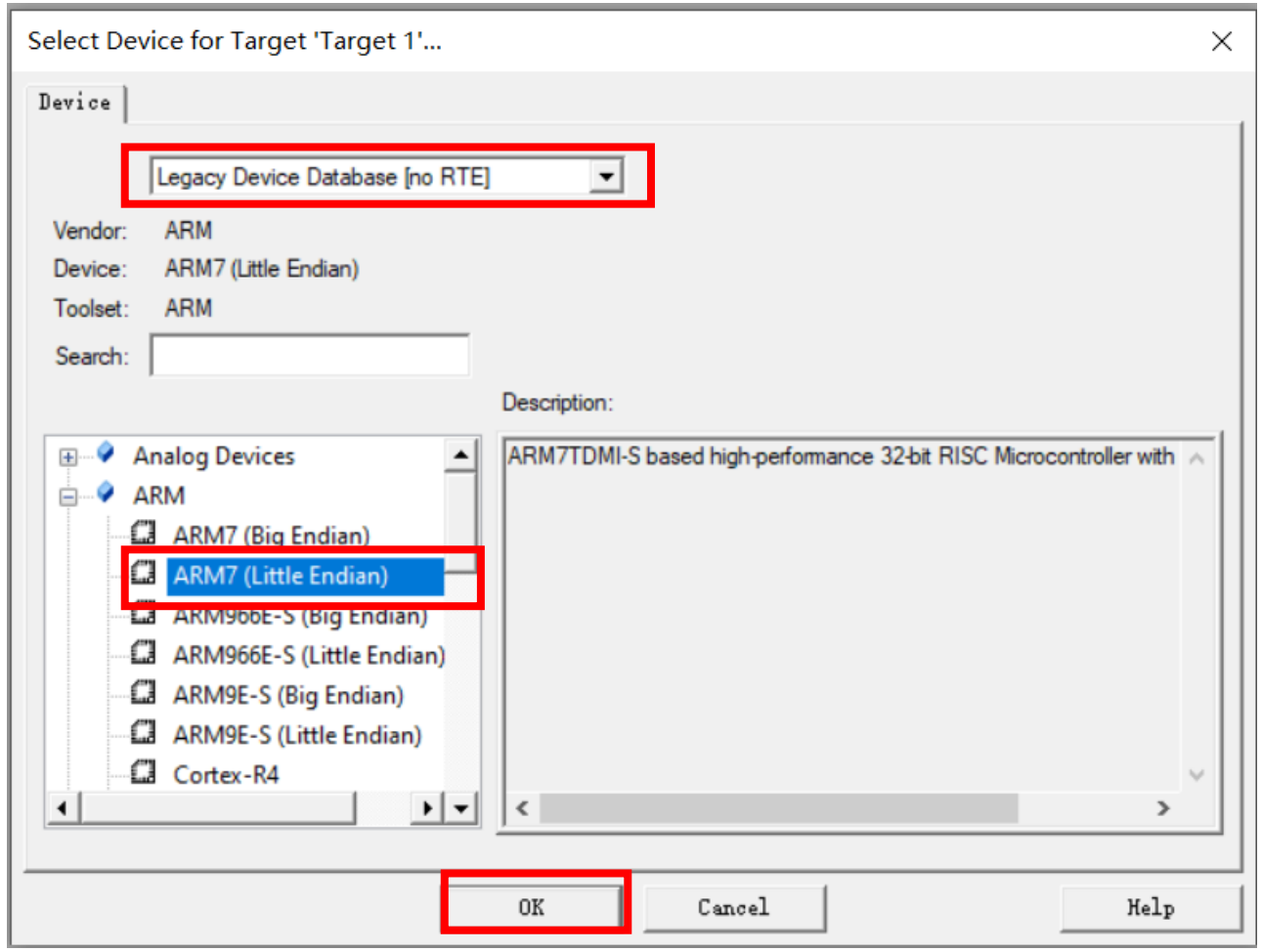
2. New a µVision project, save it to a specified location you like.





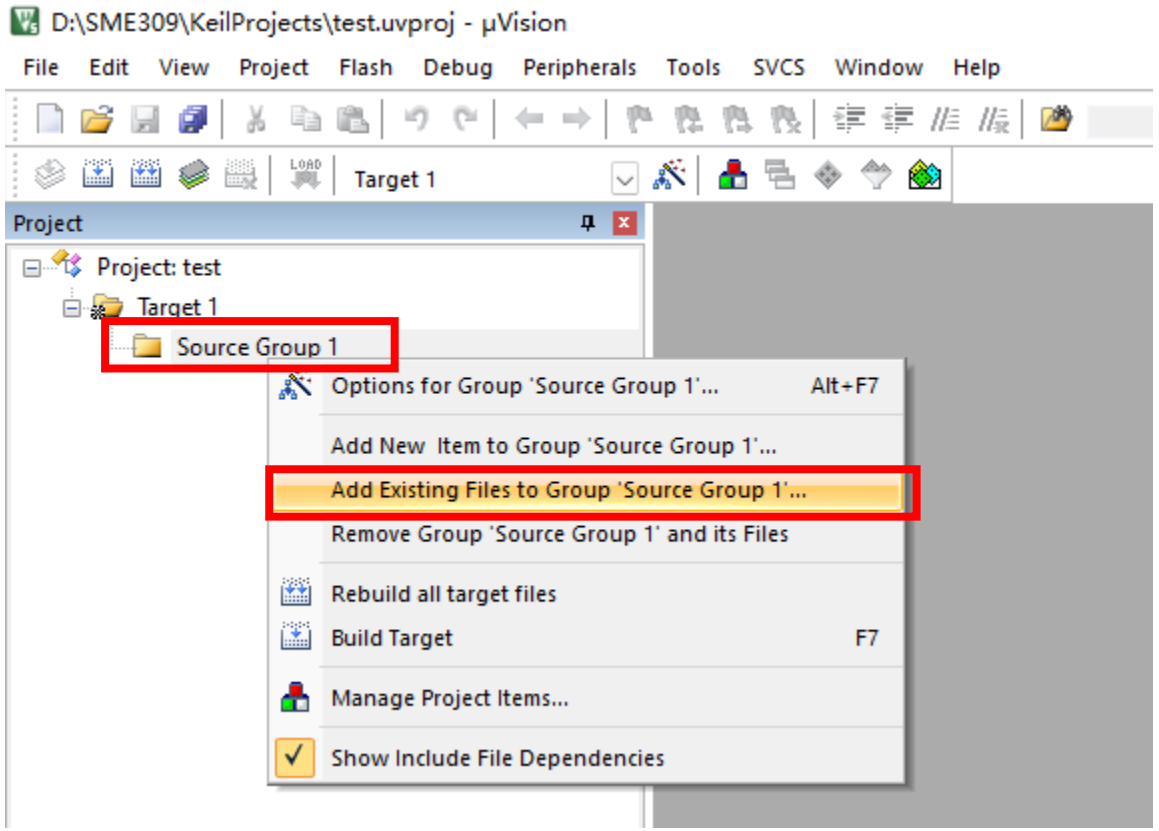
Keil Quick Start

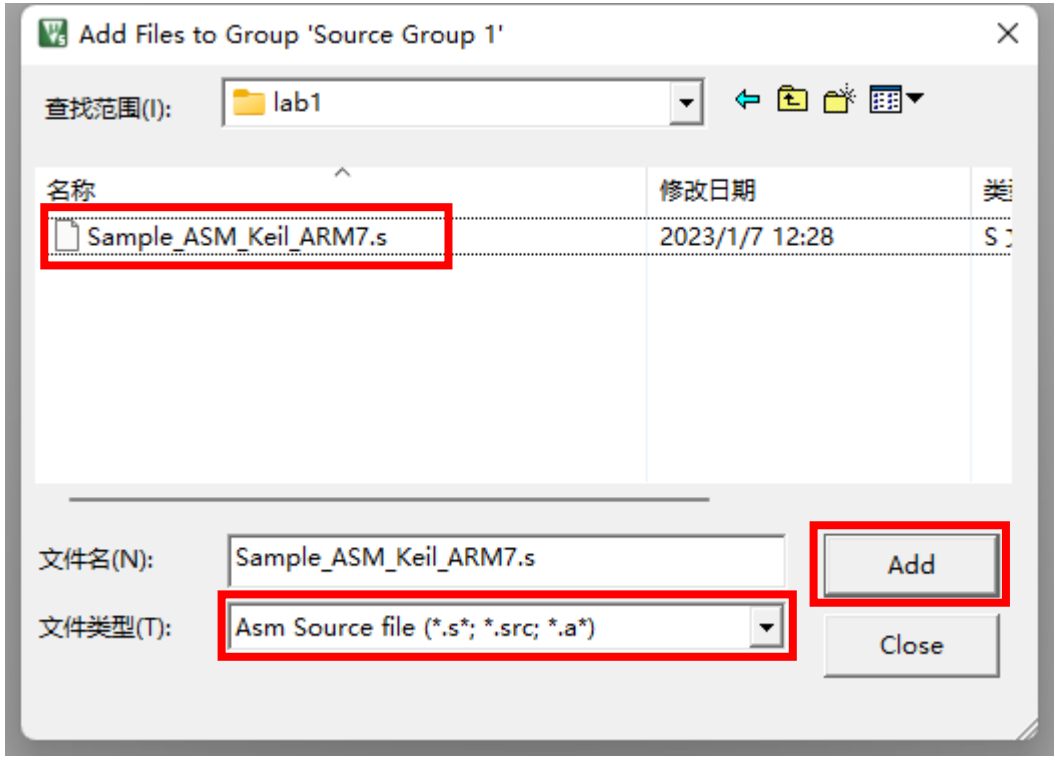
3. Choose the device “ARM7 (Little Endian)”, and OK.



Keil Quick Start

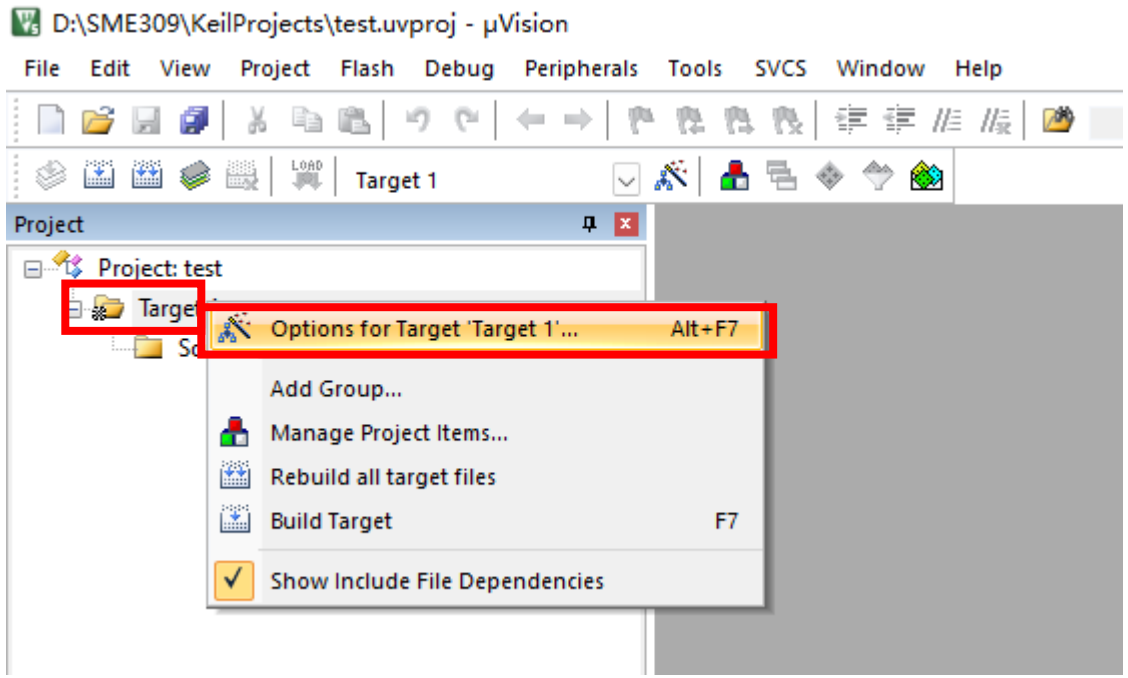
4. Add the assembly code ".s" file into the project.

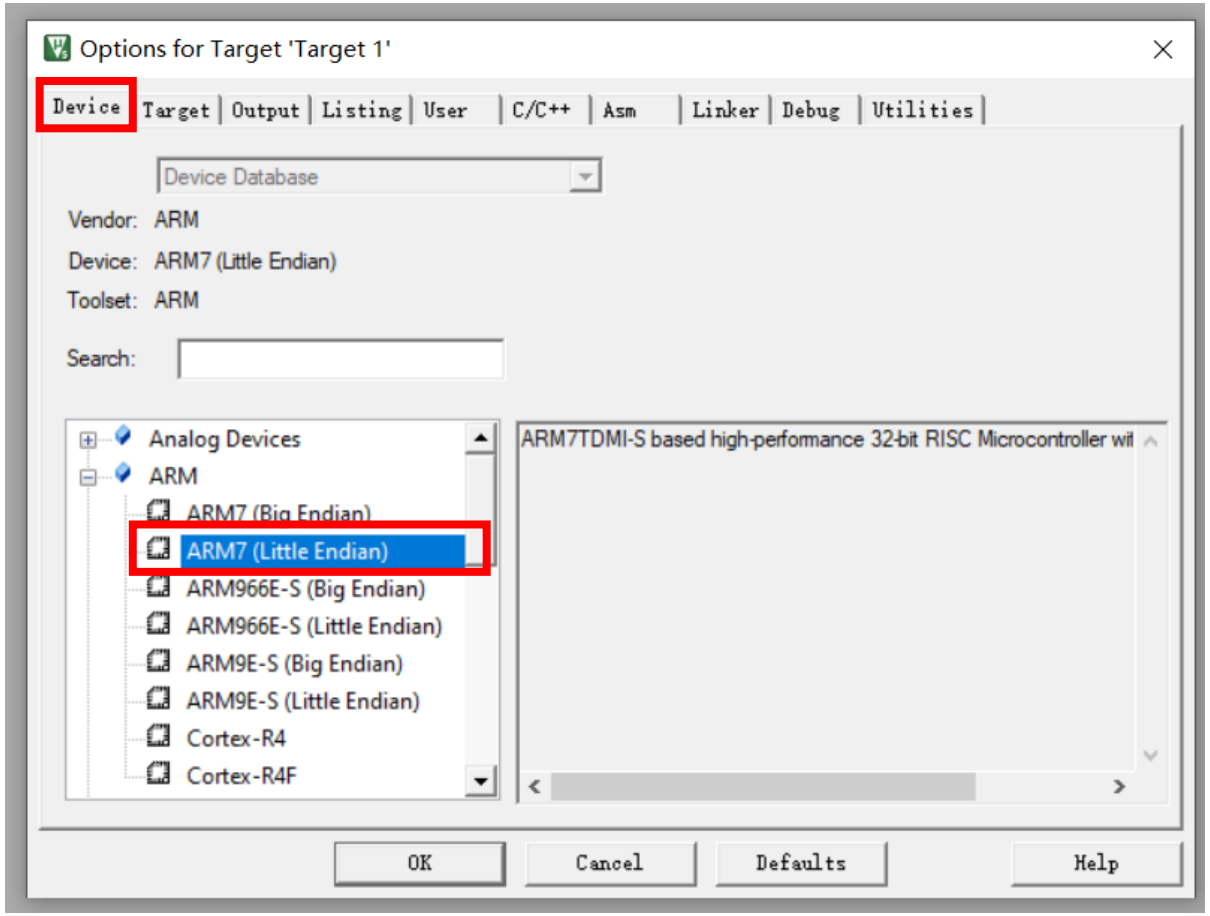




Keil Quick Start

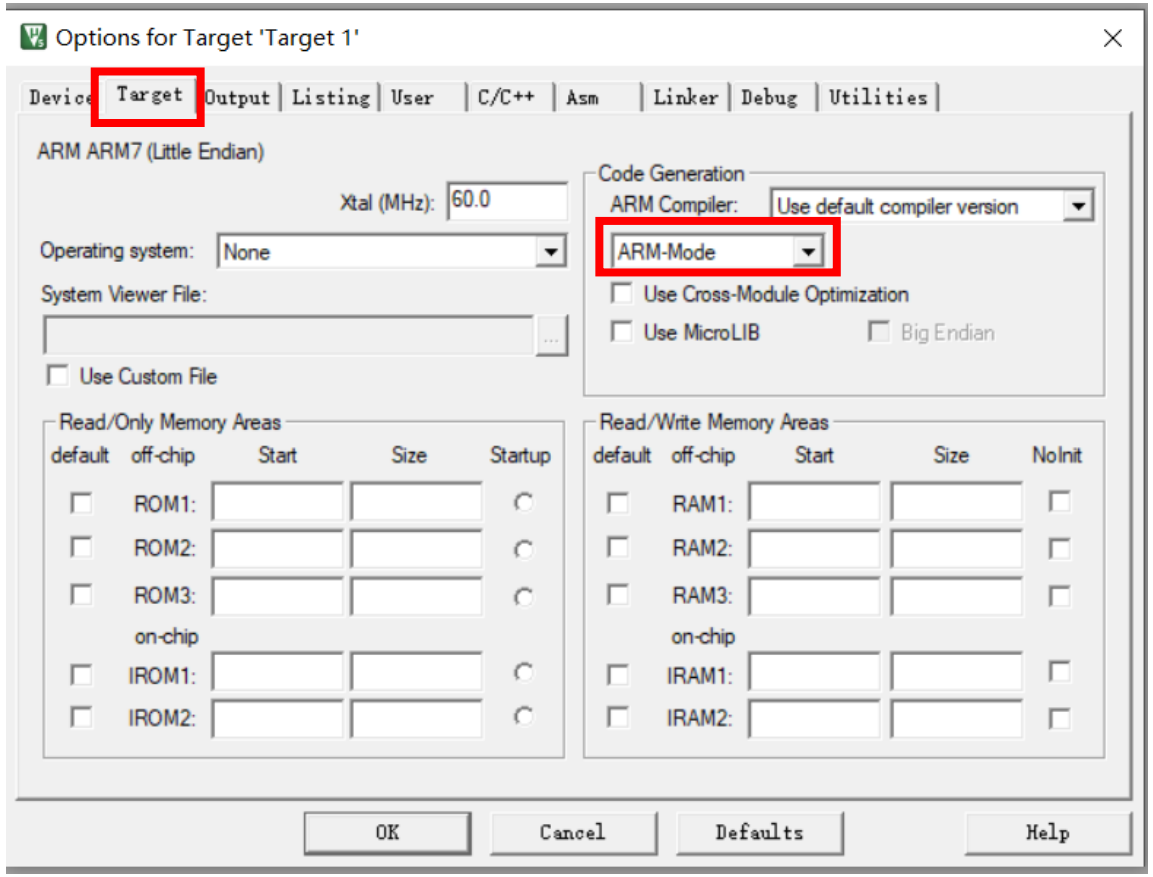
5. Check again for Target 1.

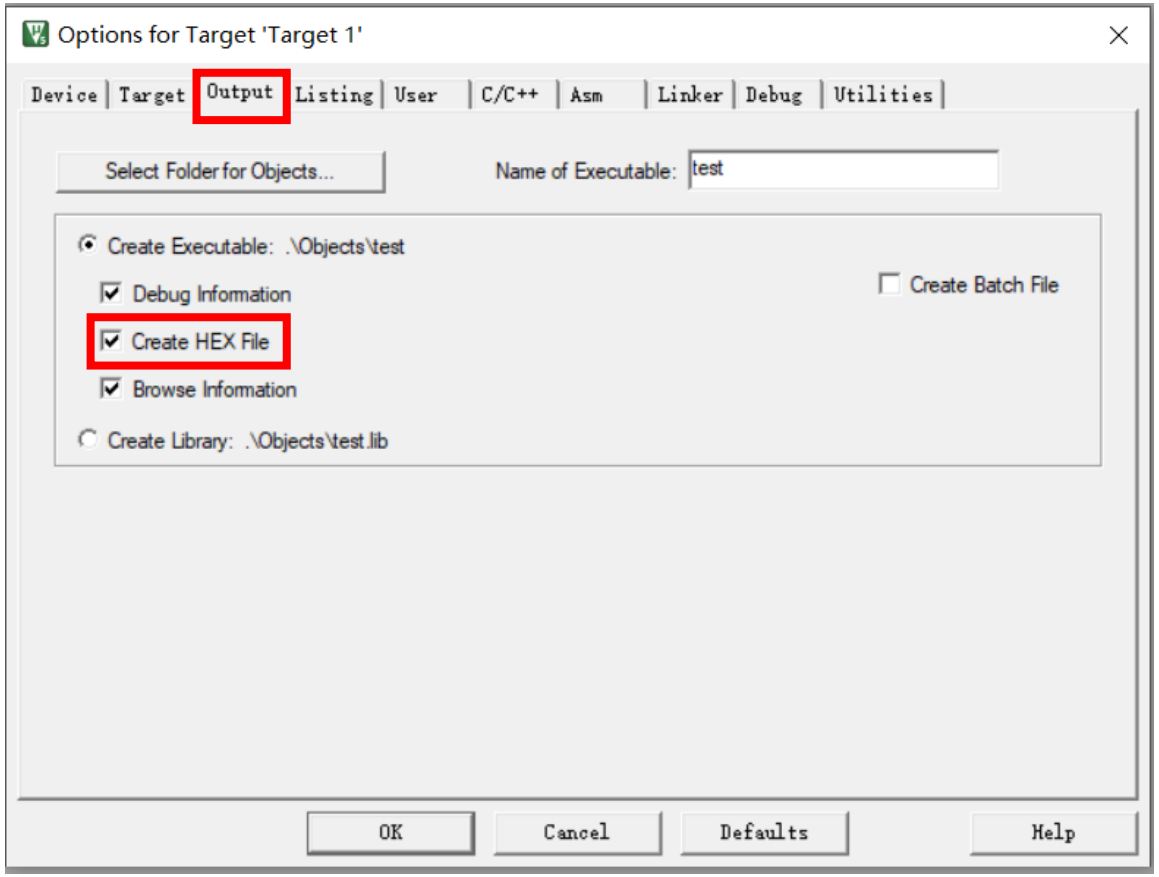




Keil Quick Start

6. Set target as ARM-Mode; set Create HEX file.



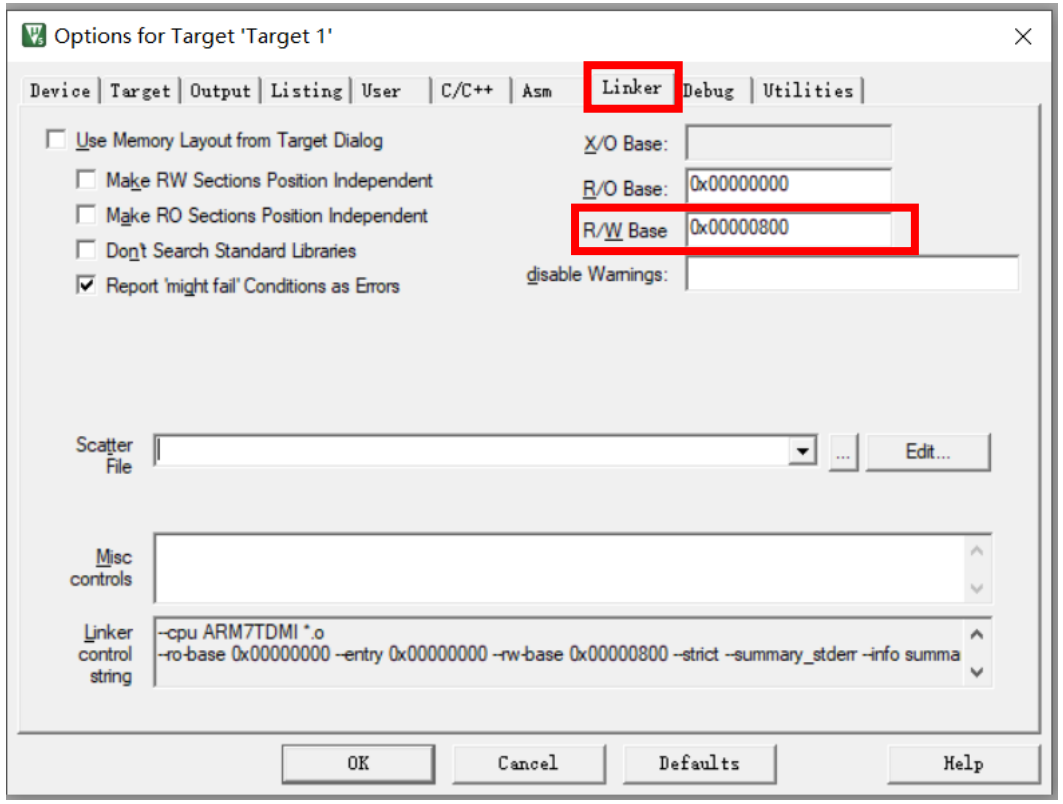


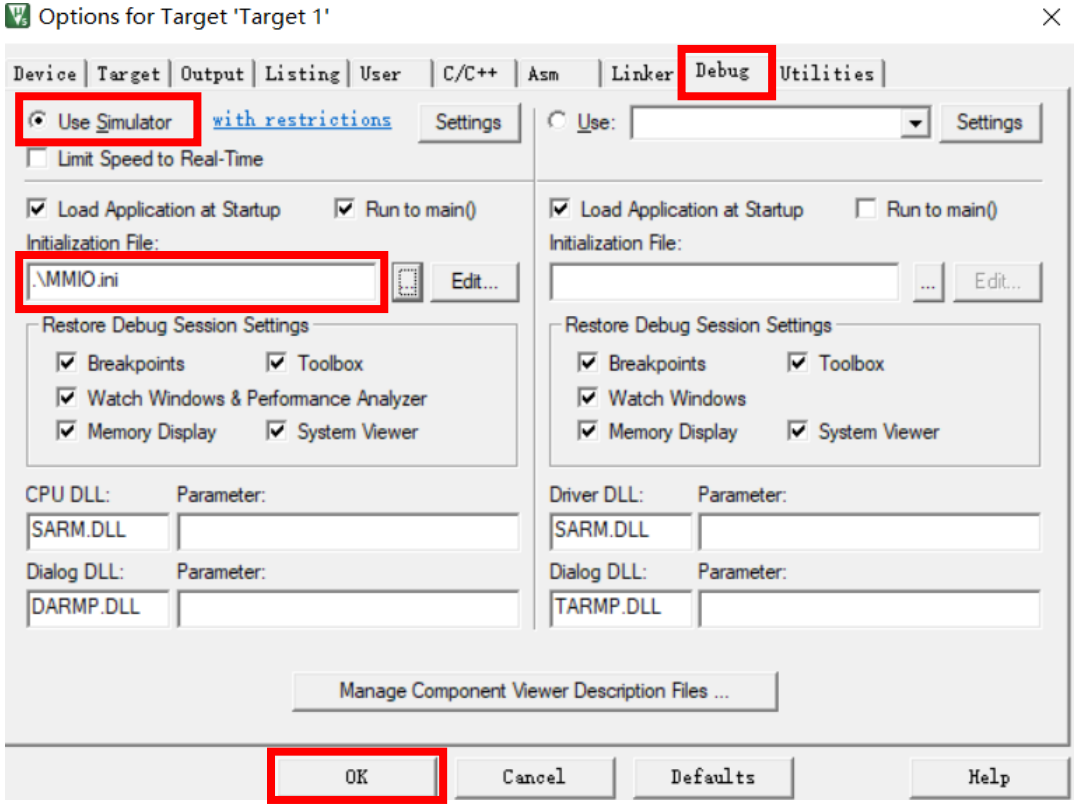
Keil Quick Start

7. Set Linker and Debug as the figures below.

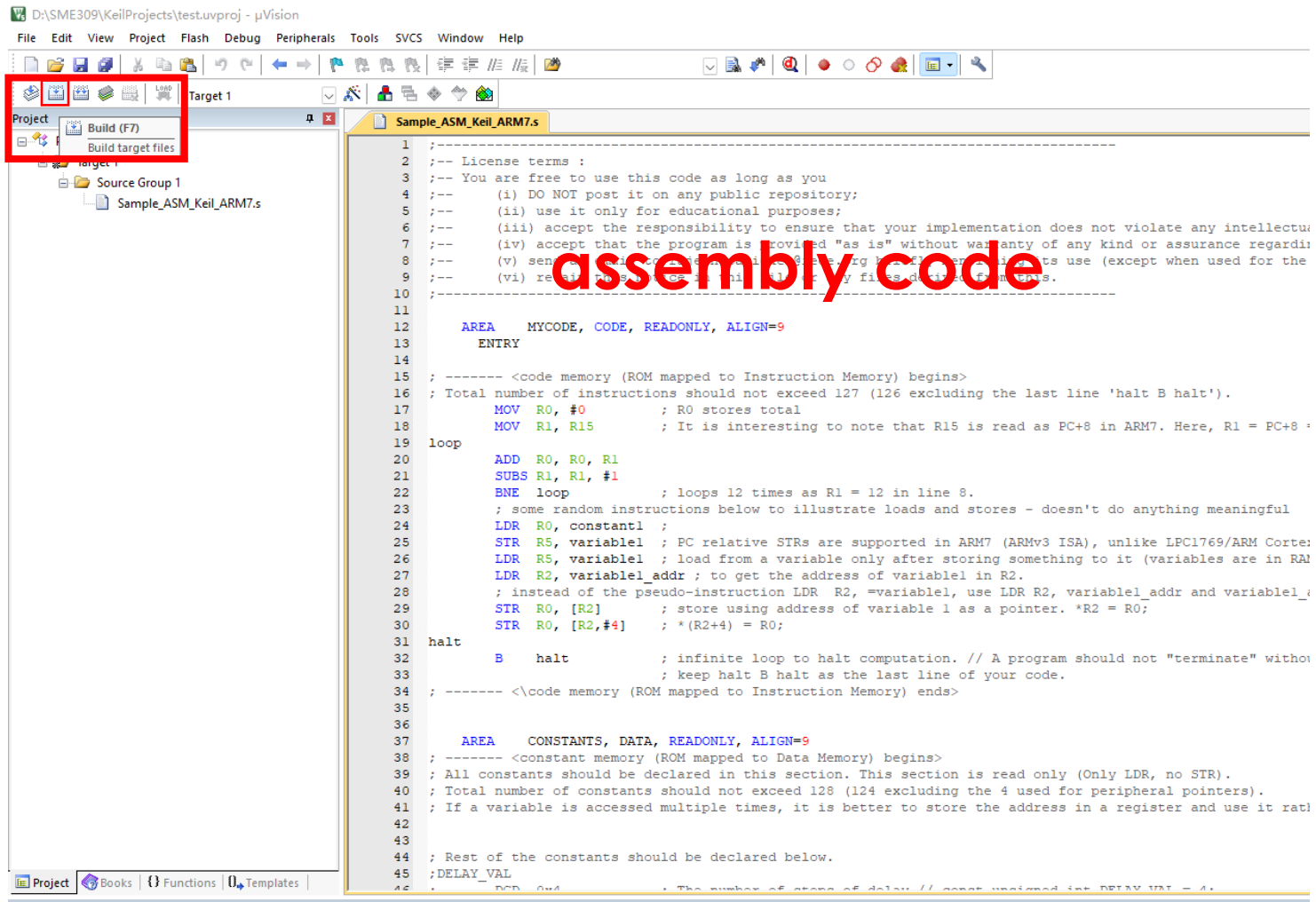
   R/W Base is the base address of Read/Write;

   Initialization file is used to allow R/W access to IO space.





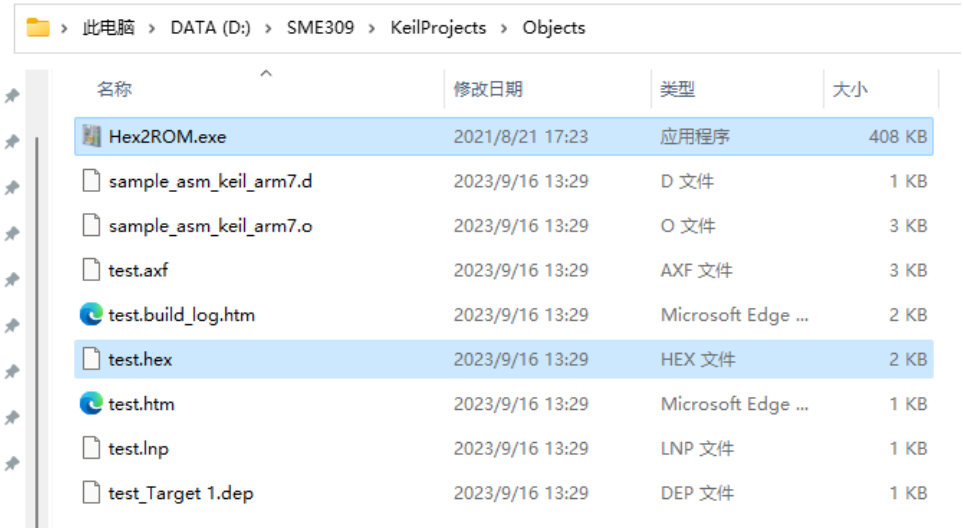
Keil Quick Start



8. Build the project.

Then you can find ".hex" file in Objects folder.

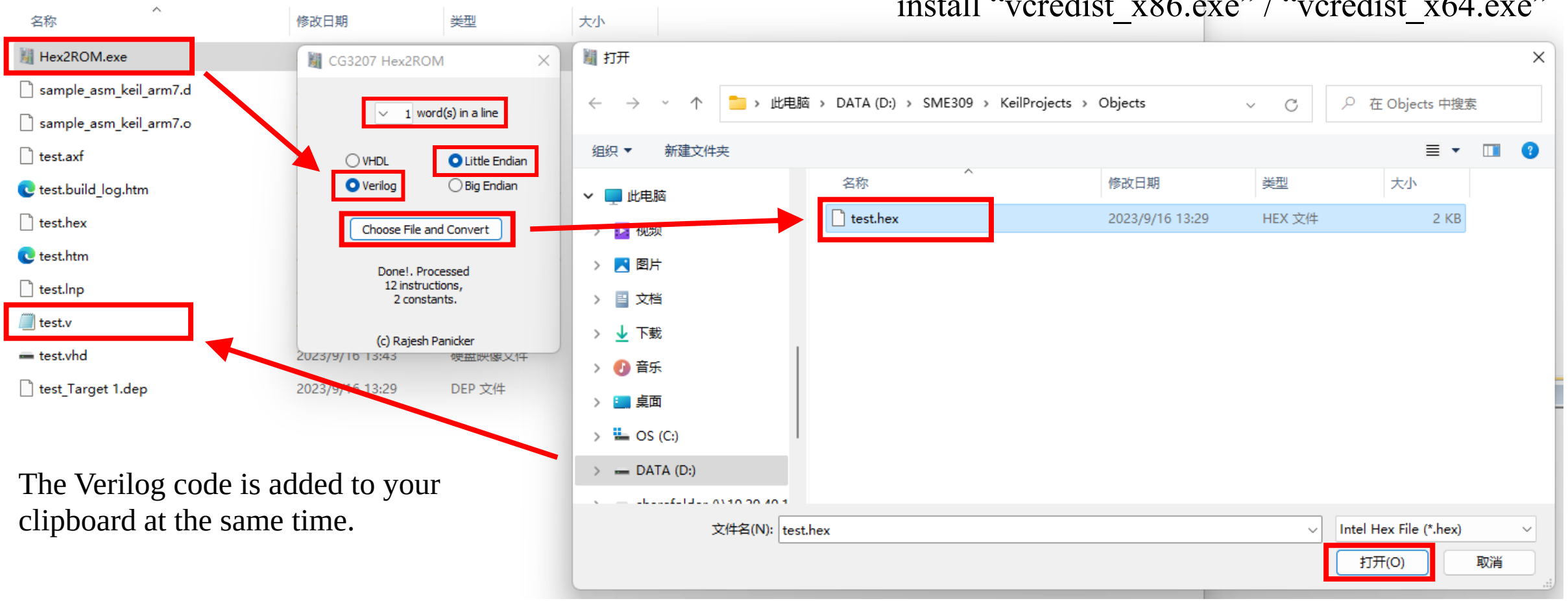
Copy the "Hex2ROM.exe" file into this folder



Keil Quick Start

9. Convert HEX file to Verilog code.

If ".dll" files are missing, download and install "vcredist_x86.exe" / "vcredist_x64.exe"



The Verilog code is added to your clipboard at the same time.

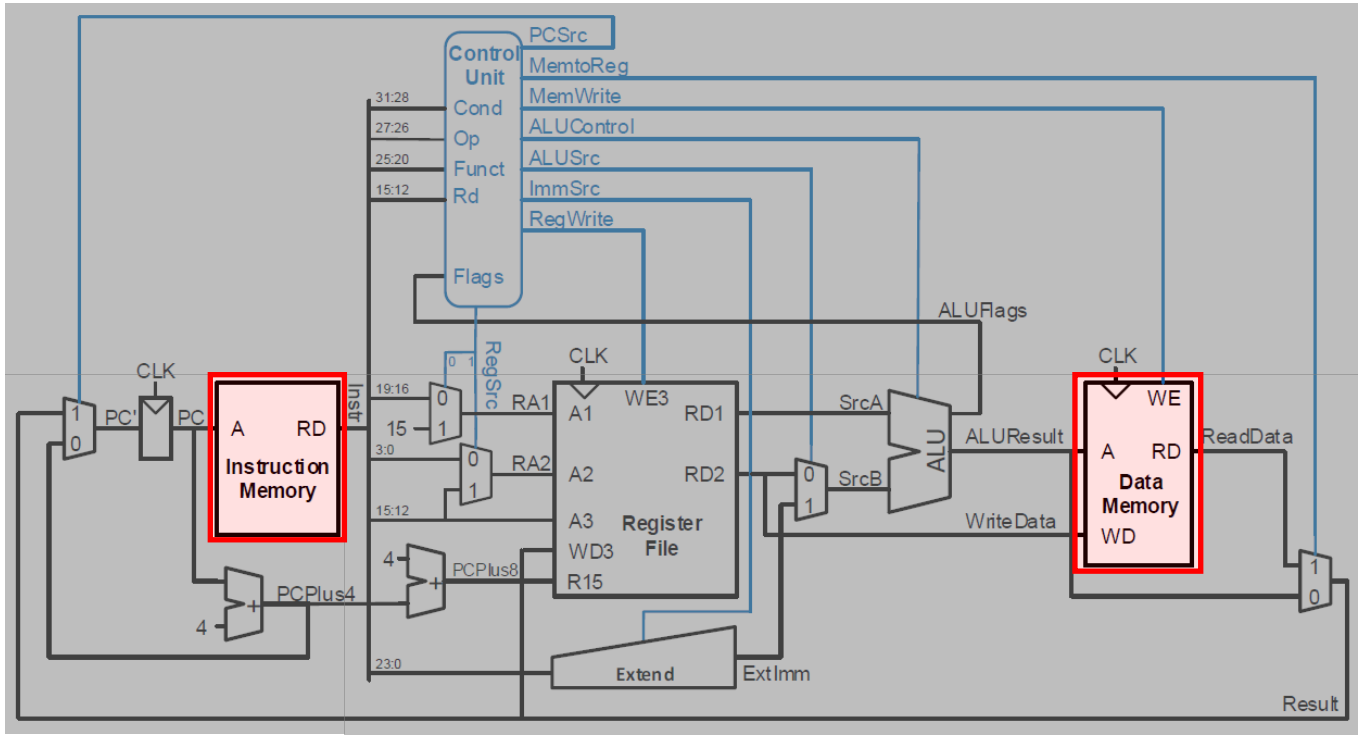
Keil Quick Start

```
//------------------------------------------------------------
// Instruction Memory
//------------------------------------------------------------
initial begin
        INSTR_MEM[0] = 32'hE3A00000;
        INSTR_MEM[1] = 32'hE1A0100F;
        INSTR_MEM[2] = 32'hE0800001;
        INSTR_MEM[3] = 32'hE2511001;
        INSTR_MEM[4] = 32'h1AFFFFFC;
        INSTR_MEM[5] = 32'hE59F01E8;
        INSTR_MEM[6] = 32'hE58F57E0;
        INSTR_MEM[7] = 32'hE59F57DC;
        INSTR_MEM[8] = 32'hE59F21D8;
        INSTR_MEM[9] = 32'hE5820000;
        INSTR_MEM[10] = 32'hE5820004;
        INSTR_MEM[11] = 32'hEAFFFFFE;
        for(i = 12; i < 128; i = i+1) begin
            INSTR_MEM[i] = 32'h0;
        end
end
```

machine code

```
//------------------------------------------------------------
// Data (Constant) Memory
//------------------------------------------------------------
initial begin
        DATA_CONST_MEM[0] = 32'h00000800;
        DATA_CONST_MEM[1] = 32'hABCD1234;
        for(i = 2; i < 128; i = i+1) begin
            DATA_CONST_MEM[i] = 32'h0;
        end
end
```

10. Check the output Verilog file, which is used to initialize Instruction and Data memory of your CPU.



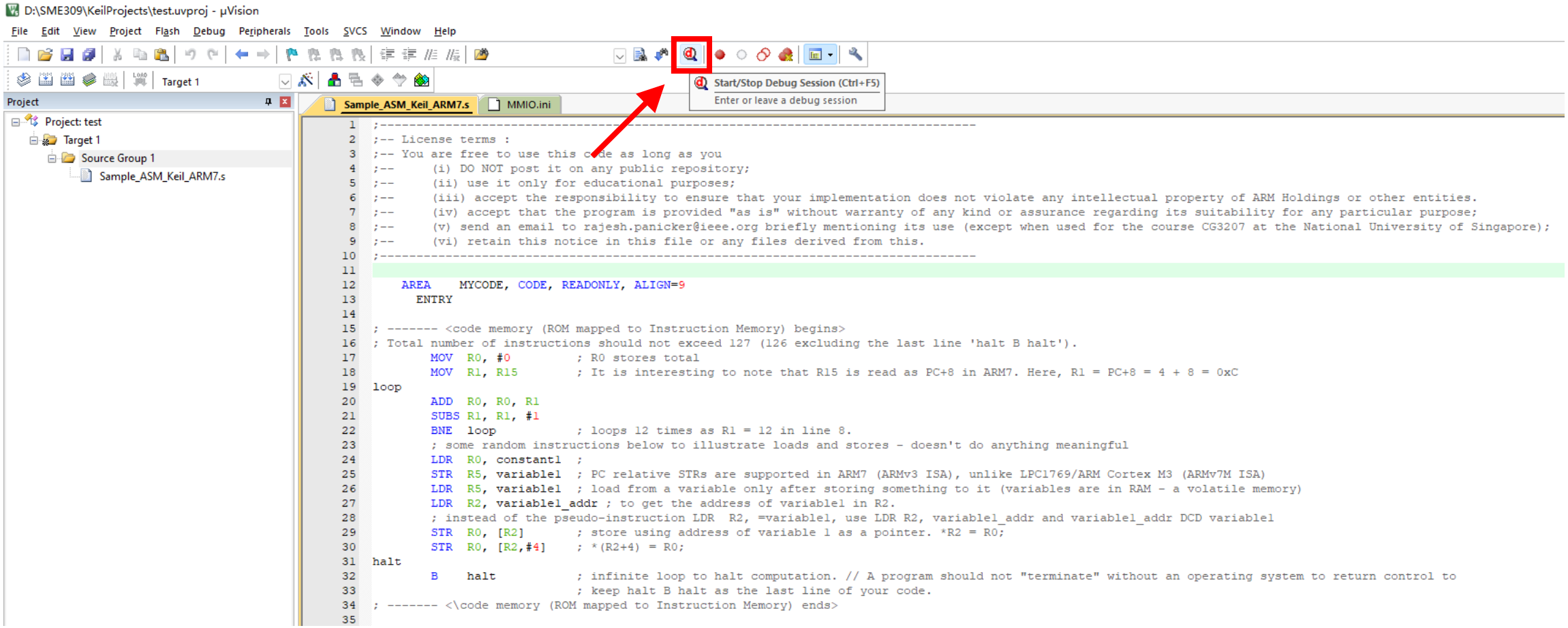
Content

SUSTech Southern University of Science and Technology

1. Background and introduction
2. Keil Quick Start
3. **Debug with Keil**
4. Seven-Segment Display
5. Vivado Tips
6. Lab1 Task

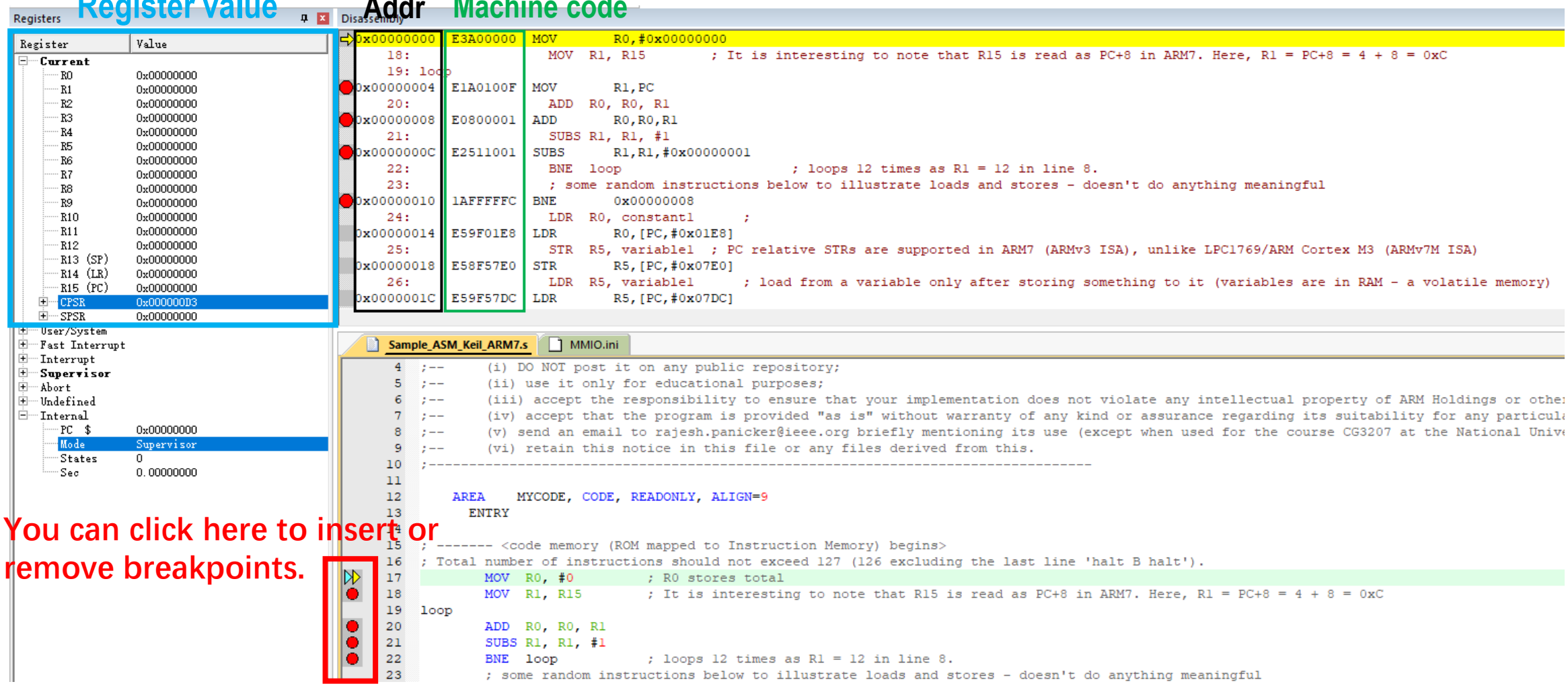
Debug with Keil

1. Start Debug Session .



Debug with Keil

2. Current values of Registers is shown on the left.

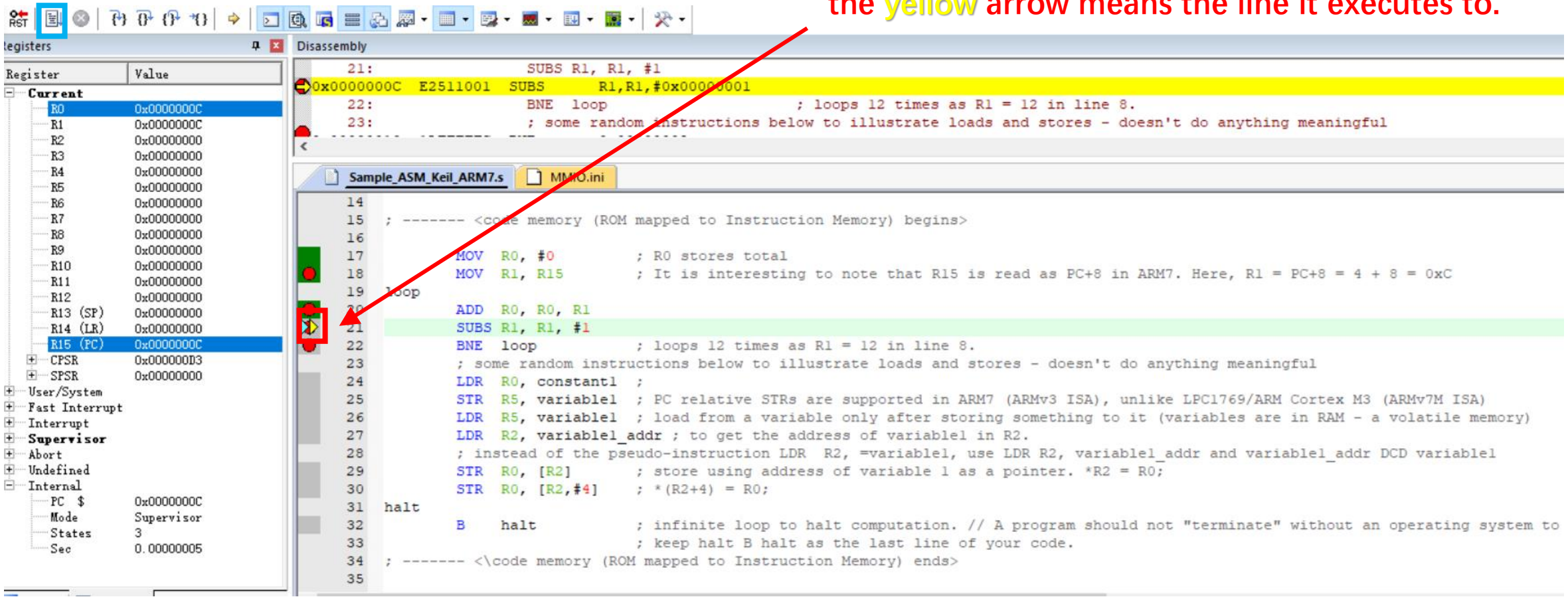


Debug with Keil

3. Run and stop.

You can click here to run your program , and it will stop at the breakpoint.

The blue arrow means the line you choose, and the yellow arrow means the line it executes to.

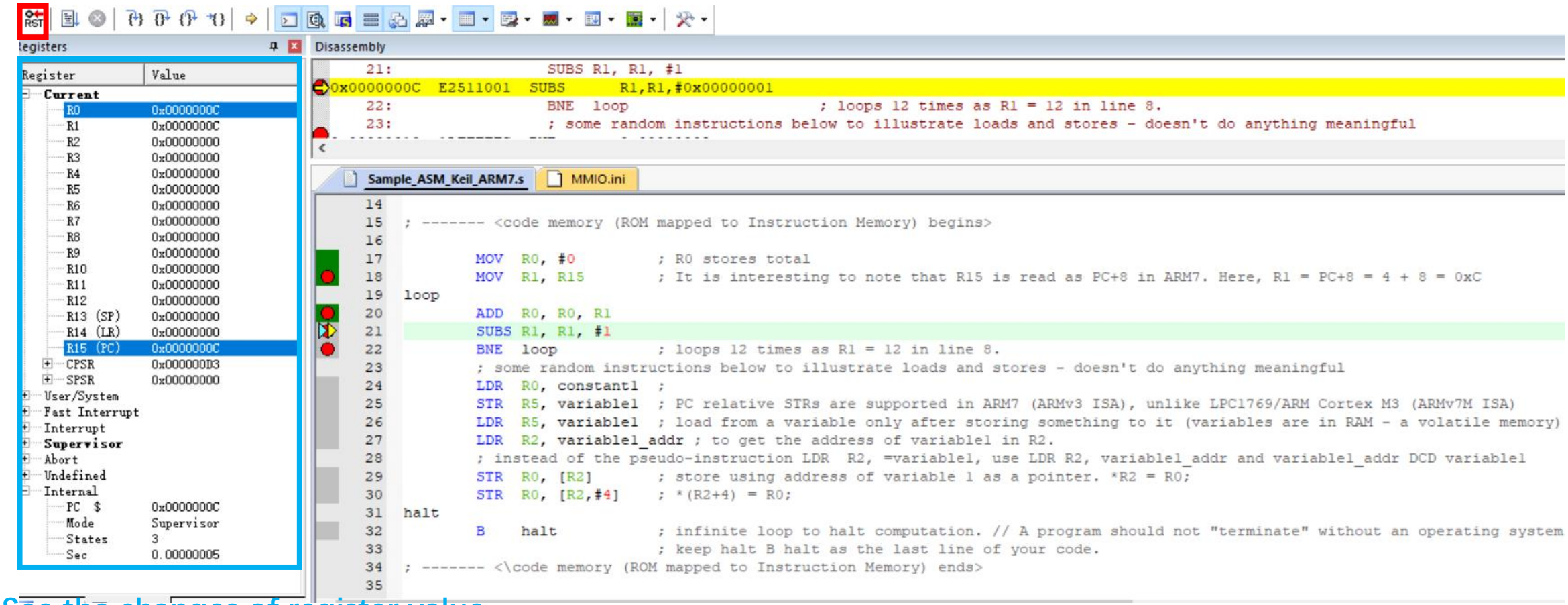


Debug with Keil

4. Reset.

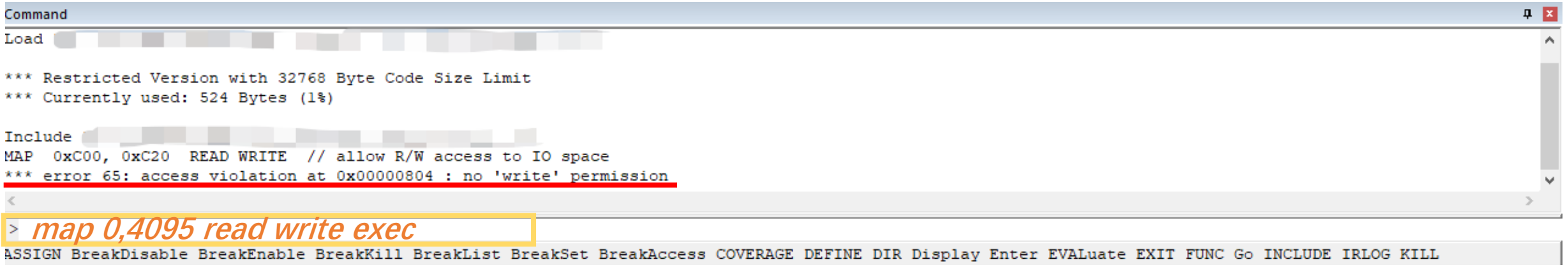
Click here to reset

PAY ATTENTION : your memory may not reset by click this button.
(try exiting debug mode and entering it again)



See the changes of register value to help you debug

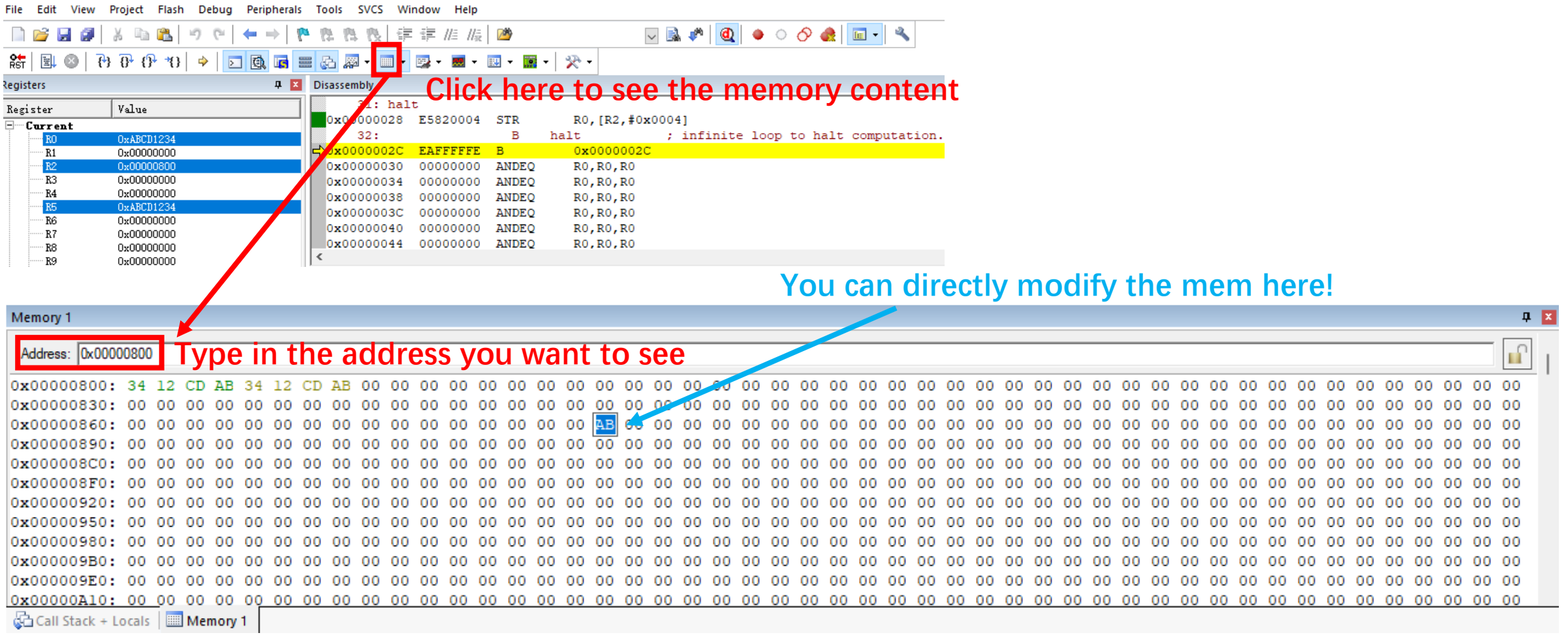
Debug with Keil



If you see this error when debugging, try typing 'map 0,4095 read write exec' on the command window, then restart simulation.

Debug with Keil

5. Memory.

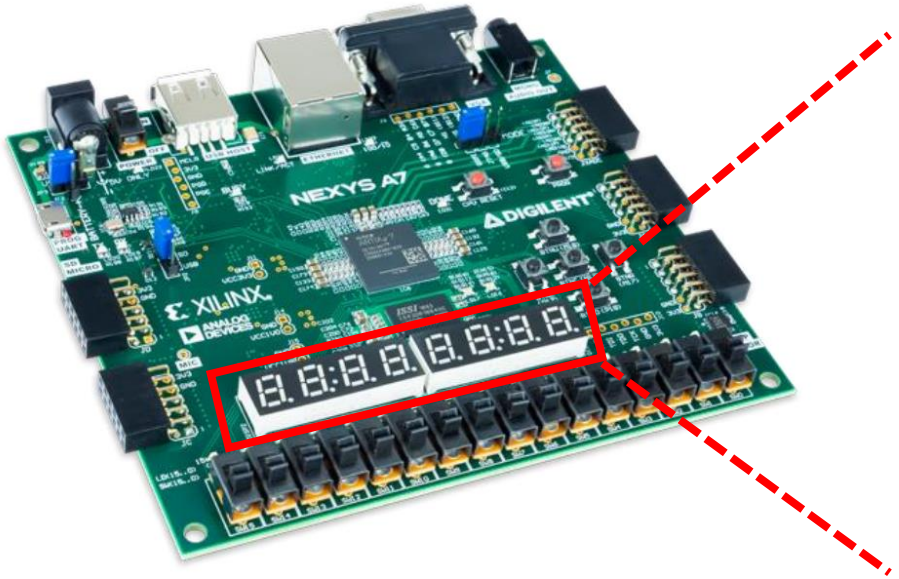


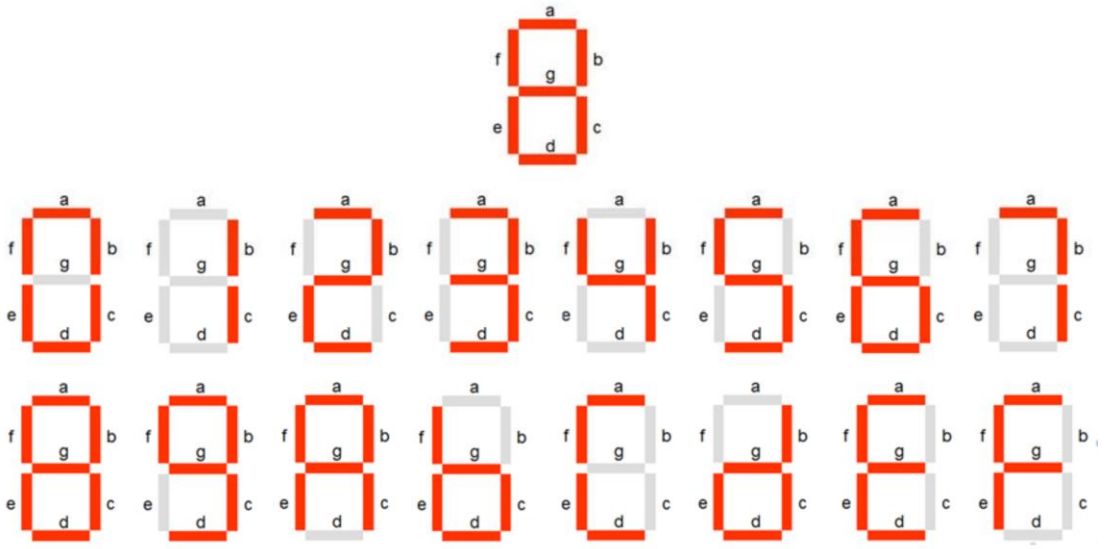
Content

SUSTech Southern University of Science and Technology

1. Background and introduction
2. Keil Quick Start
3. Debug with Keil
4. **Seven-Segment Display**
5. Vivado Tips
6. Lab1 Task

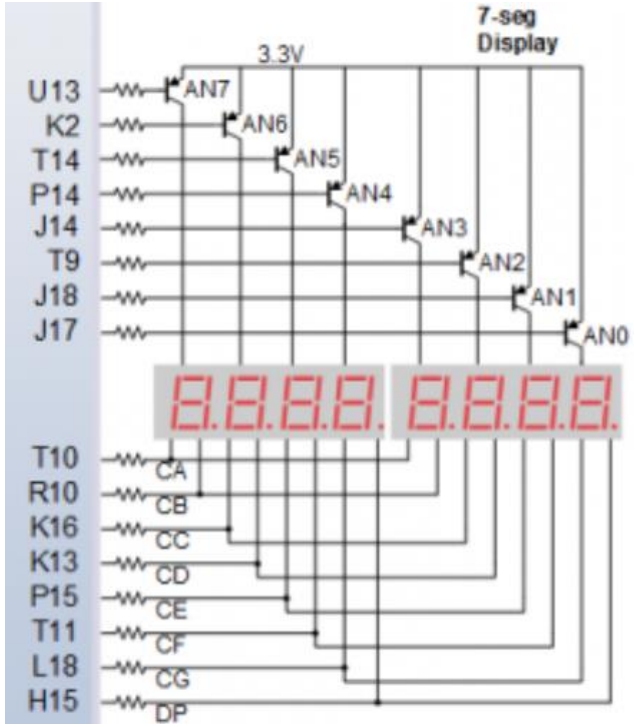
Seven-Segment Display

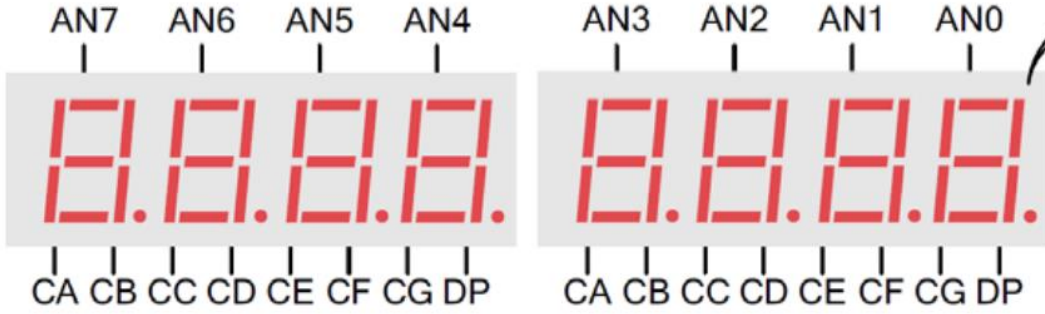




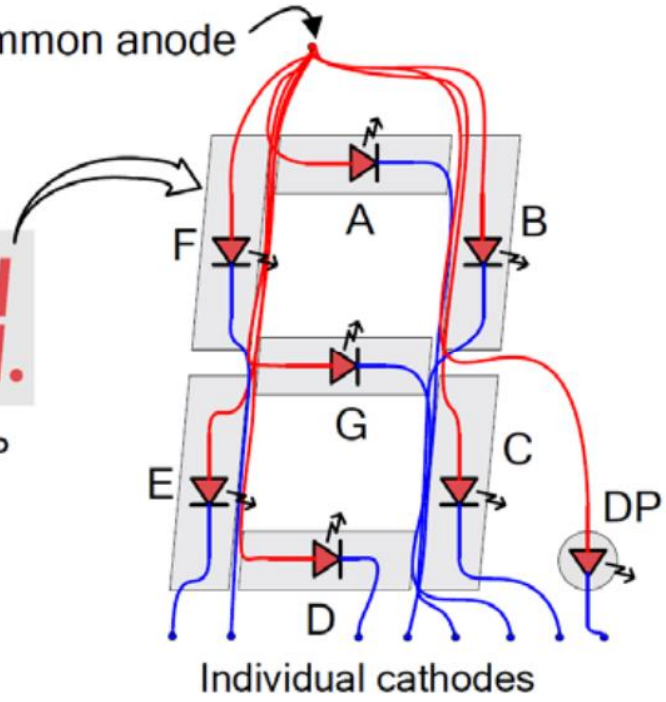
The Nexys A7 board contains two four-digit **common anode seven-segment** LED displays, configured to behave like **a single eight-digit display**. Each of the eight digits is composed of seven segments arranged in a “figure 8” pattern, with an LED embedded in each segment. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark.

Seven-Segment Display





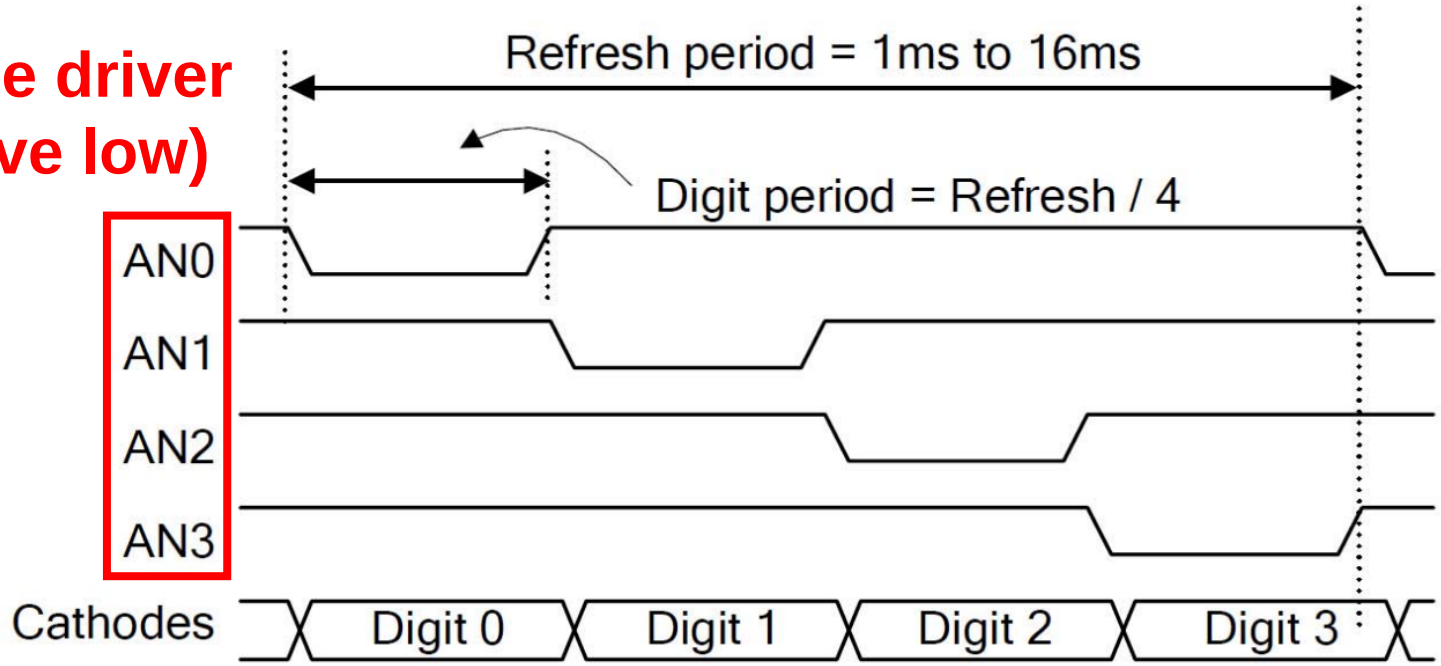
Eight-digit Seven Segment Display

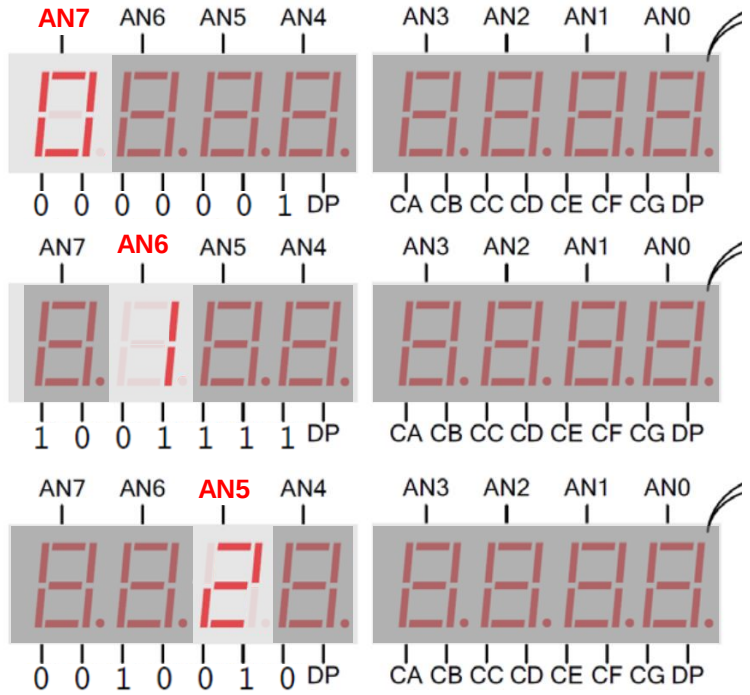


The anodes of the seven LEDs forming each digit are tied together into one "**common anode**" circuit node through a **PNP transistor** to +3.3V, as shown in the above figure, but the LED cathodes remain separate. A control **signal of 0 to a cathode** will turn on the LED segment and a signal of 1 will turn it off.

Seven-Segment Display

Anode driver (active low)





. . . . . .

A scanning display controller circuit can be used to show an eight-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession at an update rate that is **faster than the human eye can detect**. Each digit is illuminated just one-eighth of the time.

Seven-Segment Display

x	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
B	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
D	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

1 = off

0 = on

The seven cathode signals are available as inputs to the 8-digit display. This signal connection scheme creates a multiplexed display, where the **cathode signals are common to all digits** but they can only illuminate the segments of the digit whose corresponding anode signal is asserted.

The table shows output cathode values for each segment ca – cg needed to display all hex values form 0 – F.

Refer to *Nexys A7™ FPGA Board Reference Manual (page 23-24)* for more detailed information.

How to use 7-segment display

The output port for 7-segment display of top module can be mapped like this

anode (8)	OUT		☑	(Multiple)	LVCMOS18	1.800	12	SLOW	NONE
anode[7]	OUT	U13	☑	14	LVCMOS18	1.800	12	SLOW	NONE
anode[6]	OUT	K2	☑	35	LVCMOS18	1.800	12	SLOW	NONE
anode[5]	OUT	T14	☑	14	LVCMOS18	1.800	12	SLOW	NONE
anode[4]	OUT	P14	☑	14	LVCMOS18	1.800	12	SLOW	NONE
anode[3]	OUT	J14	☑	15	LVCMOS18	1.800	12	SLOW	NONE
anode[2]	OUT	T9	☑	14	LVCMOS18	1.800	12	SLOW	NONE
anode[1]	OUT	J18	☑	15	LVCMOS18	1.800	12	SLOW	NONE
anode[0]	OUT	J17	☑	15	LVCMOS18	1.800	12	SLOW	NONE
cathode (7)	OUT		☑	(Multiple)	LVCMOS18	1.800	12	SLOW	NONE
cathode[6]	OUT	L18	☑	14	LVCMOS18	1.800	12	SLOW	NONE
cathode[5]	OUT	T11	☑	14	LVCMOS18	1.800	12	SLOW	NONE
cathode[4]	OUT	P15	☑	14	LVCMOS18	1.800	12	SLOW	NONE
cathode[3]	OUT	K13	☑	15	LVCMOS18	1.800	12	SLOW	NONE
cathode[2]	OUT	K16	☑	15	LVCMOS18	1.800	12	SLOW	NONE
cathode[1]	OUT	R10	☑	14	LVCMOS18	1.800	12	SLOW	NONE
cathode[0]	OUT	T10	☑	14	LVCMOS18	1.800	12	SLOW	NONE
dp	OUT	H15	☑	15	LVCMOS18	1.800	12	SLOW	NONE

Content

SUSTech Southern University of Science and Technology

1. Background and introduction
2. Keil Quick Start
3. Debug with Keil
4. Seven-Segment Display
5. **Vivado Tips**
6. Lab1 Task

Vivado Tip1: A testbench example

```
`timescale 1ns / 1ps
module control_tb();
    reg clk;
    reg pause;
    reg speedup;
    wire [8:0] addr;

    control ctrl1(clk, pause, speedup, addr);

    initial begin
        clk = 0;
        forever #50 clk=~clk;
    end

    initial begin
        pause = 0;speedup = 0;
        #1650 pause = 1;
        #3200 pause = 0;
        #500 speedup = 1;
        #1600 pause = 1;
        #1600 speedup = 0; pause = 0;
        #3200 $finish;
        end

endmodule
```

Set time scale

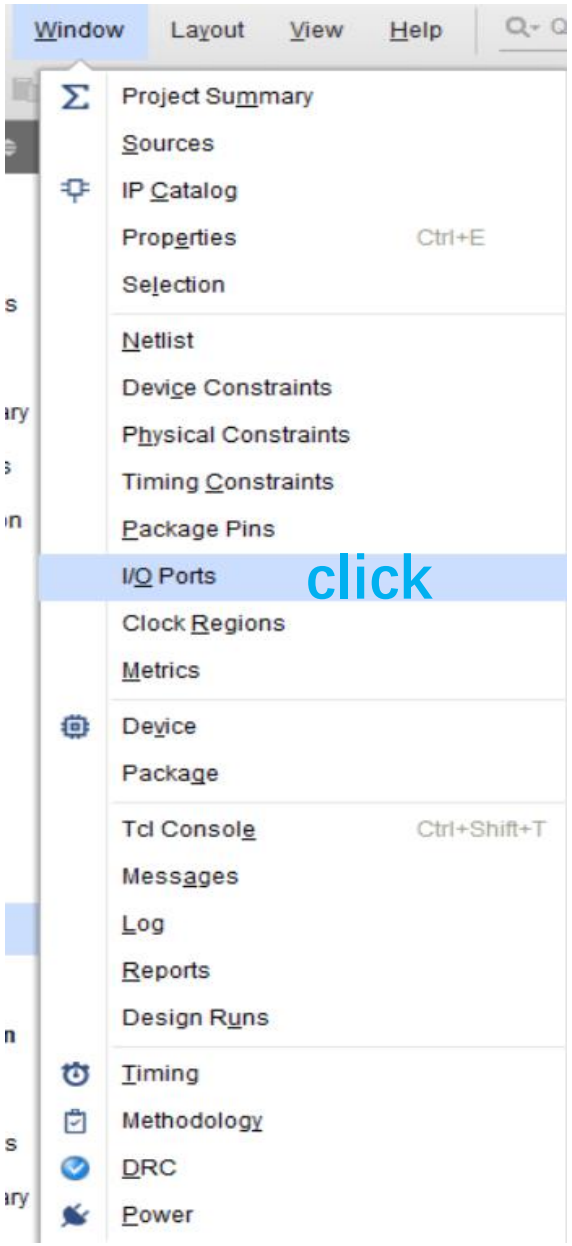
Reg: the input of your DUT
Wire: the output of your DUT

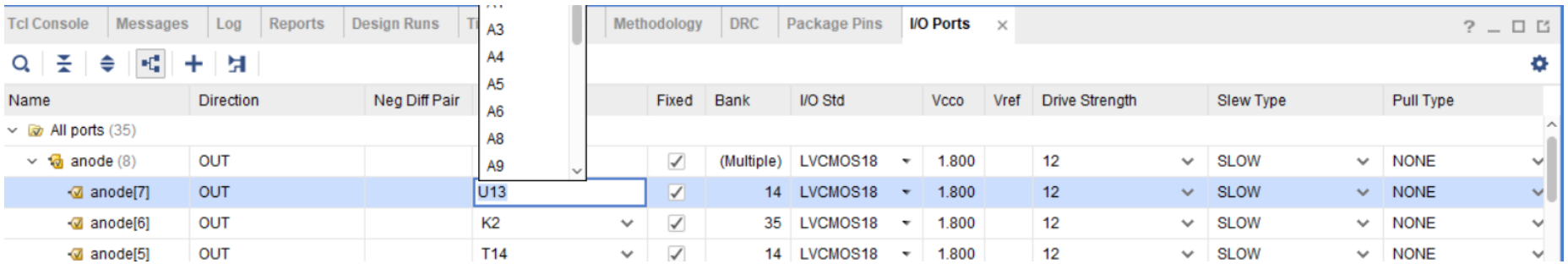
Module instantiation

Initialize clk

Generate stimuli
($finish is used to finish the simulation)

Vivado Tip2: How to generate a constraint

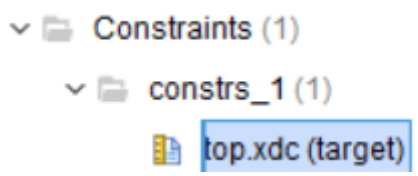




I/O Ports window will show on the bottom. Then choose package pins to map your port in your module.

Then Ctrl+S, a constraint file will be generated.

You can also directly create a constraint file, and typing into it! (some syntax should be followed, you can search it on the internet)

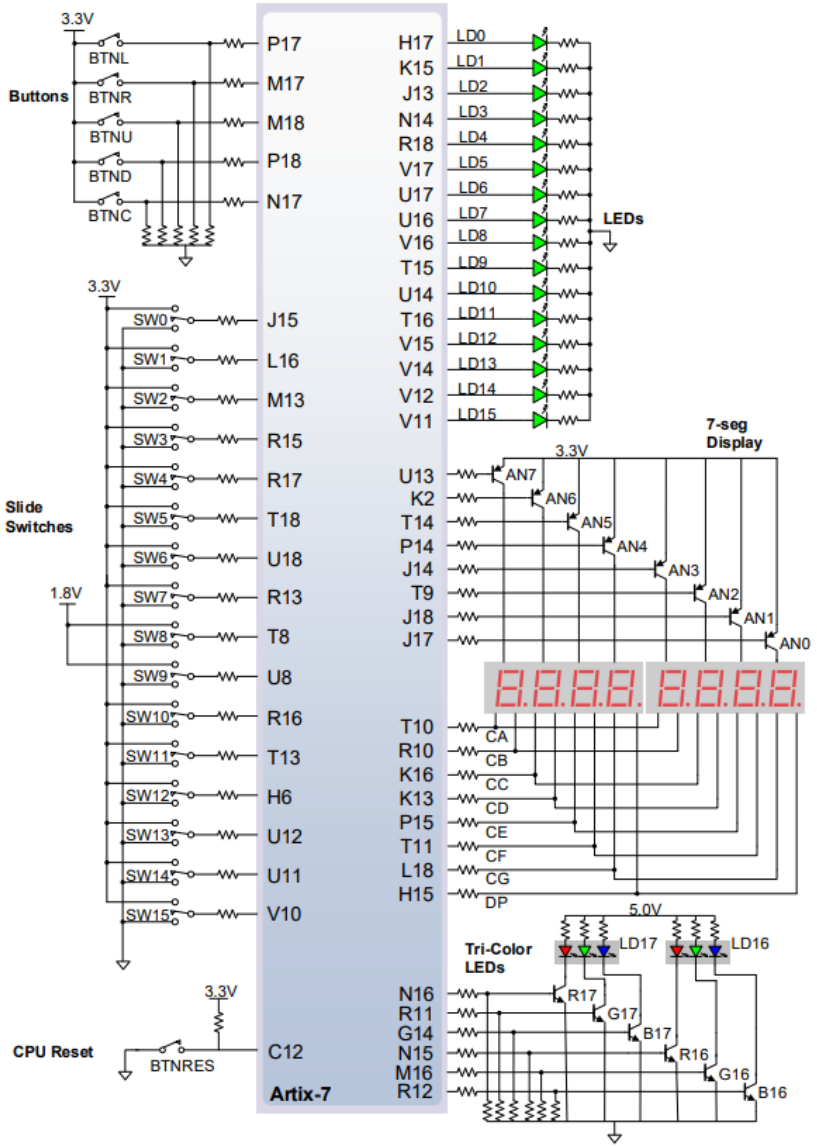


Module ports

```
1 set_property PACKAGE_PIN H17 [get_ports {led_data[0]}]
2 set_property PACKAGE_PIN K15 [get_ports {led_data[1]}]
3 set_property PACKAGE_PIN J13 [get_ports {led_data[2]}]
4 set_property PACKAGE_PIN N14 [get_ports {led_data[3]}]
5 set_property PACKAGE_PIN R18 [get_ports {led_data[4]}]
6 set_property PACKAGE_PIN V17 [get_ports {led_data[5]}]
7 set_property PACKAGE_PIN V11 [get_ports {led_data[15]}]
```

Physical pins

Vivado Tip2: How to generate a constraint



If you have no idea which pin should be mapped to your port, please refer to *Nexys A7™ FPGA Board Reference Manual* (*nexys-a7_rm.pdf* in our QQ group).

If there are some problems when using this FPGA board, please also refer to *Nexys A7™ FPGA Board Reference Manual*.

Another tip: download the board.xdc file on Github, uncomment the block you used, and just need to modify the ports.
https://github.com/Digilent/digilent-xdc

Vivado Tip3: Initialize your ROM

```
initial begin
//instruction memory initial
instr_mem[0] = 32'd0;
instr_mem[1] = 32'd1;
instr_mem[2] = 32'd2;
instr_mem[3] = 32'd3;
instr_mem[4] = 32'd4;
instr_mem[5] = 32'd5;
instr_mem[6] = 32'd6;
instr_mem[7] = 32'd7;
instr_mem[8] = 32'd8;
instr_mem[9] = 32'd9;
instr_mem[10] = 32'd10;
instr_mem[11] = 32'd11;
instr_mem[12] = 32'd12;
instr_mem[13] = 32'd13;
instr_mem[14] = 32'd14;
instr_mem[15] = 32'd15;
instr_mem[16] = 32'd16;
instr_mem[17] = 32'd17;
instr_mem[18] = 32'd18;
instr_mem[19] = 32'd19;

for(i = 20; i < 128; i = i+1) begin
   instr_mem[i] = 32'd0;
   end
```

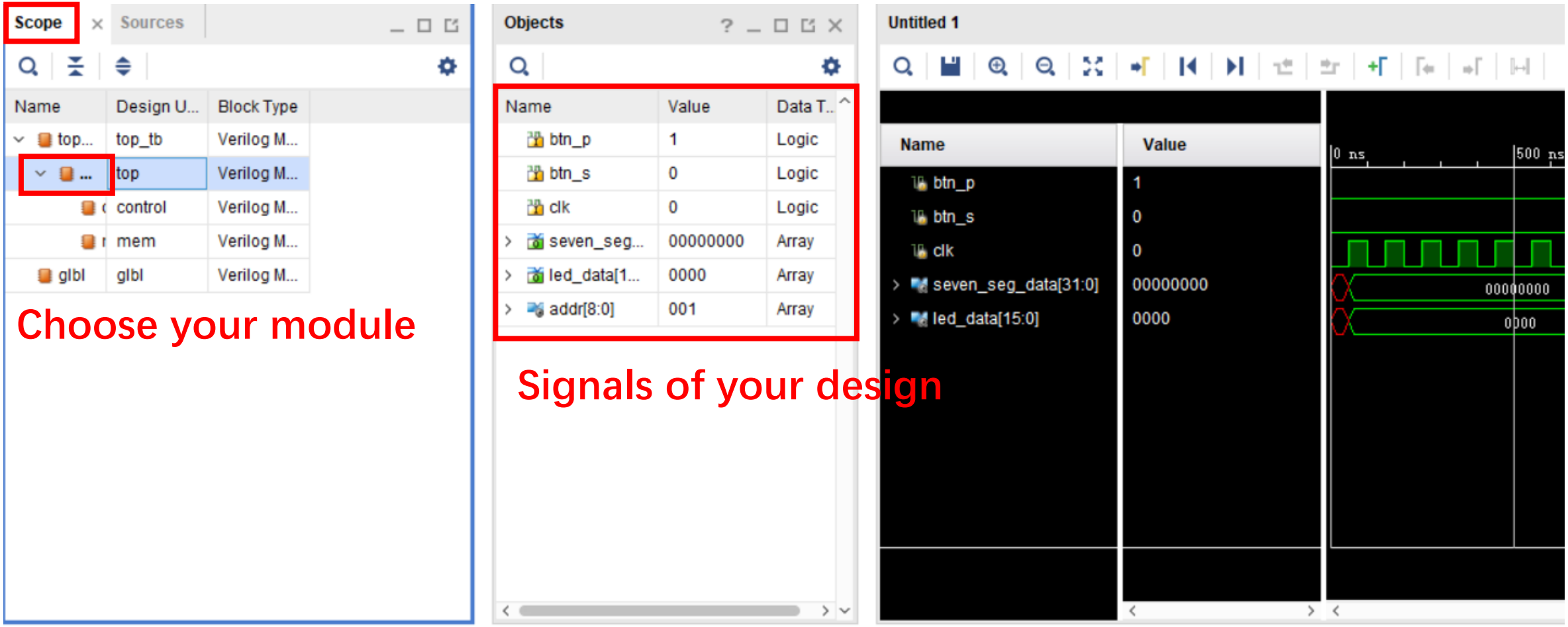
Copy the initial block into your memory module.

The initial content of ROM will be given at checkpoint. Before that, make your own initial content to test your design.

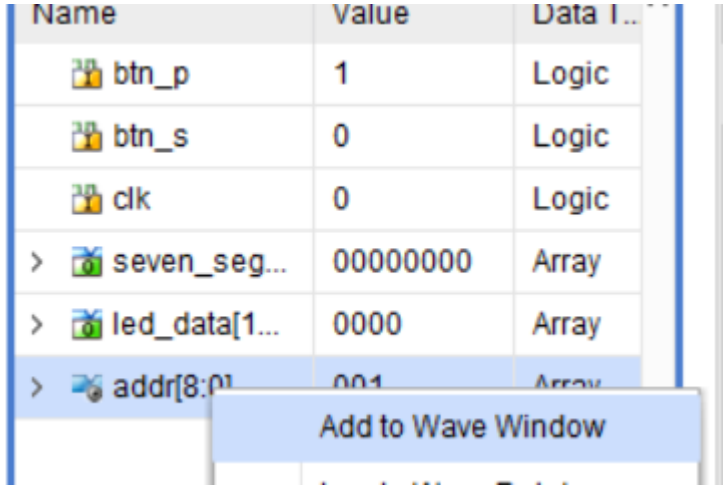
```
integer i;
reg [31:0] INSTR_MEM [127:0];
reg [31:0] DATA_CONST_MEM [127:0];

//-----------------------------------------------------------------
// Instruction Memory
//-----------------------------------------------------------------
initial begin
    INSTR_MEM[0] = 32'hE3A00000;
    INSTR_MEM[1] = 32'hE1A0100F;
    INSTR_MEM[2] = 32'hE0800001;
    INSTR_MEM[3] = 32'hE2511001;
    INSTR_MEM[4] = 32'h1AFFFFFC;
    INSTR_MEM[5] = 32'hE59F01E8;
    INSTR_MEM[6] = 32'hE58F57E0;
    INSTR_MEM[7] = 32'hE59F57DC;
    INSTR_MEM[8] = 32'hE59F21D8;
    INSTR_MEM[9] = 32'hE5820000;
    INSTR_MEM[10] = 32'hE5820004;
    INSTR_MEM[11] = 32'hEAFFFFFE;
    for(i = 12; i < 128; i = i+1) begin
        INSTR_MEM[i] = 32'h0;
    end
end

//-----------------------------------------------------------------
// Data (Constant) Memory
//-----------------------------------------------------------------
initial begin
    DATA_CONST_MEM[0] = 32'h00000800;
    DATA_CONST_MEM[1] = 32'hABCD1234;
    for(i = 2; i < 128; i = i+1) begin
        DATA_CONST_MEM[i] = 32'h0;
    end
end
```

Initialize unused location to zero

Vivado Tip4: How to see a signal inside a module when doing simulation

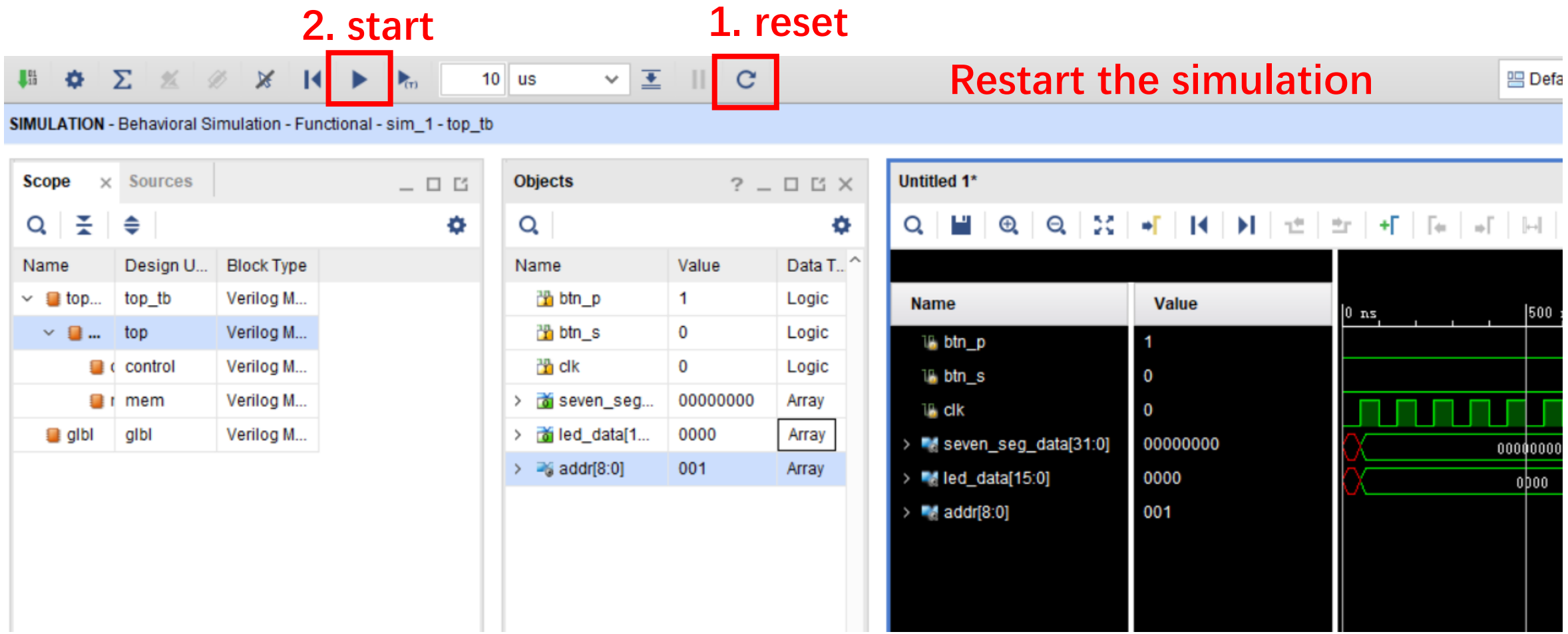


Vivado Tip4: How to see a signal inside a module when doing simulation



Right click the signal you want to see, then click ‘add to wave window’

Vivado Tip4: How to see a signal inside a module when doing simulation



Vivado Tip4: How to see a signal inside a module when doing simulation

The waveform appears!

Content

SUSTech Southern University of Science and Technology

1. Background and introduction
2. Keil Quick Start
3. Debug with Keil
4. Seven-Segment Display
5. Vivado Tips
6. **Lab1 Task**

Lab1 Task

Create an Instruction ROM (Read Only Memory) and a Data ROM, display the content of the ROMs with LED lights (for addresses) and 7-segment display (for memory content) on your FPGA board.

Requirements:

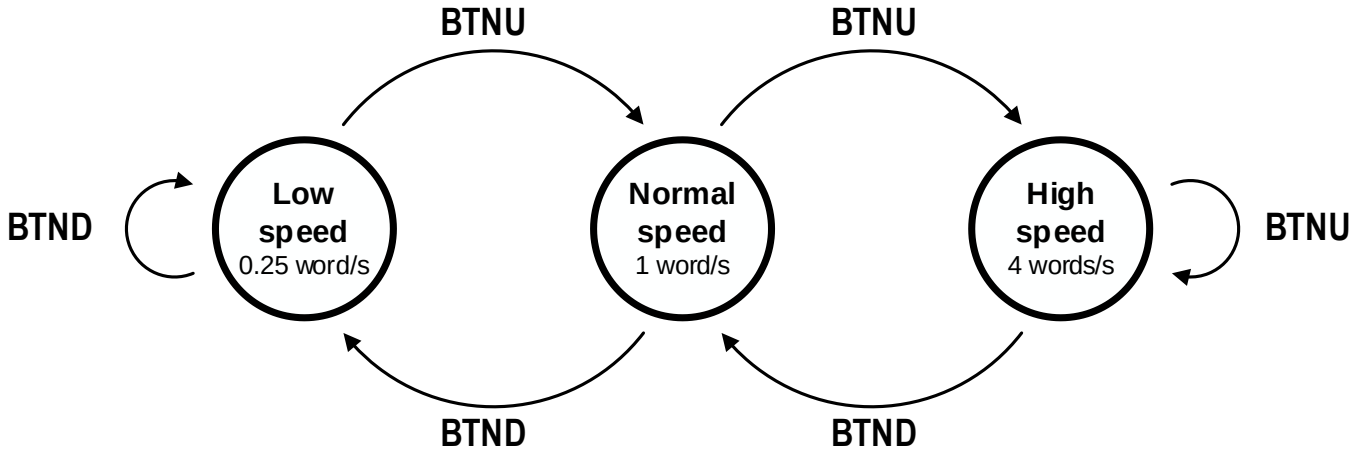
1. Your instr_mem and data_mem ROM should have a capacity of **128 words** each, and store the instructions and data respectively. Note that both memories are word addressable. There are 128 locations, each location has a content of **32 bits**, addressed with 7 bits.

2. Display the contents of Instruction and Data ROMs on the 7-segment display, and their addresses on LEDs. The speed of display should be approximately **1 instruction/datum per second**.

3. When the Instruction ROM display has completed, display the contents of Data ROM. Do this in a **cyclical manner**. If there is unused memory (does not contain useful information) in your ROM, initialize it to **zero**.

Lab1 Task

4. When **BTNC** is pressed, the display should **start and pause.** When **BTNU** is pressed, the display speed should increase to approximately **4 instructions/data per second**. When **BTND** is pressed, the display speed return to **1 instruction/datum per second**. When **BTND** is pressed again, it will speed down to **0.25 instruction/datum per second**, and pressing **BTNU** again will return to normal speed. (A simple state machine for this)



5. You should simulate all the modules in your design before programming into FPGA board, we will check your **simulation waveform** while checking all your design.
One second may be too long to simulate, you can set the max of counters to 10 in simulation.

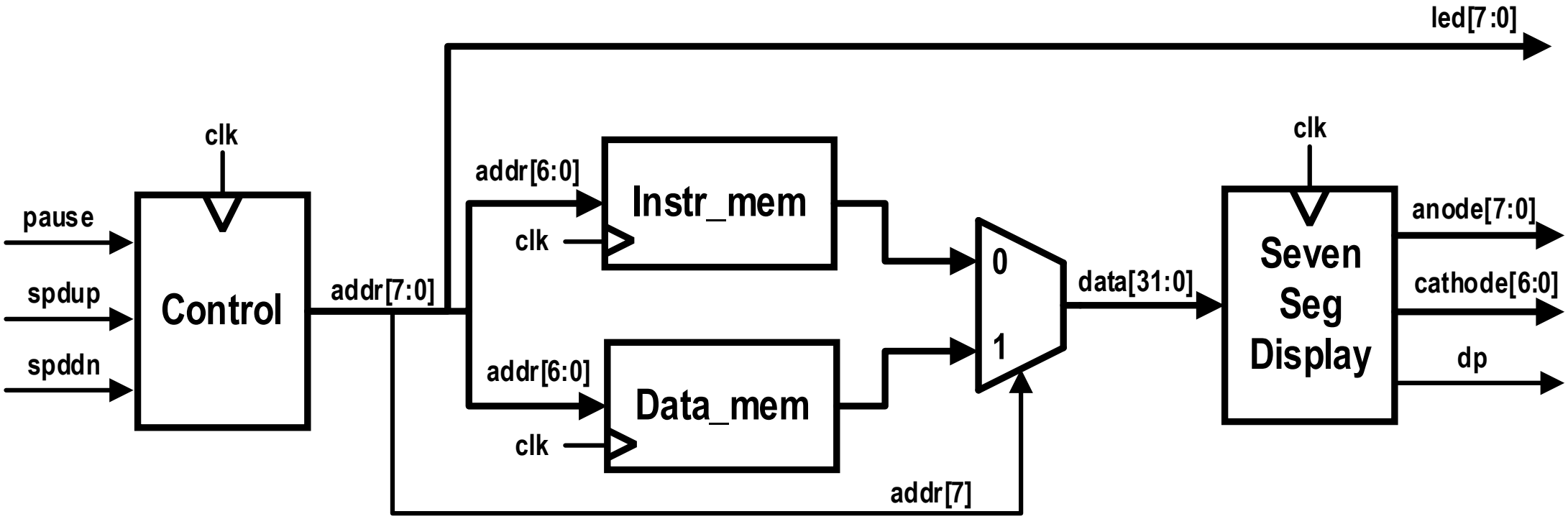
Lab1 Task

6. The **I/O port of top module** of your design should be like this

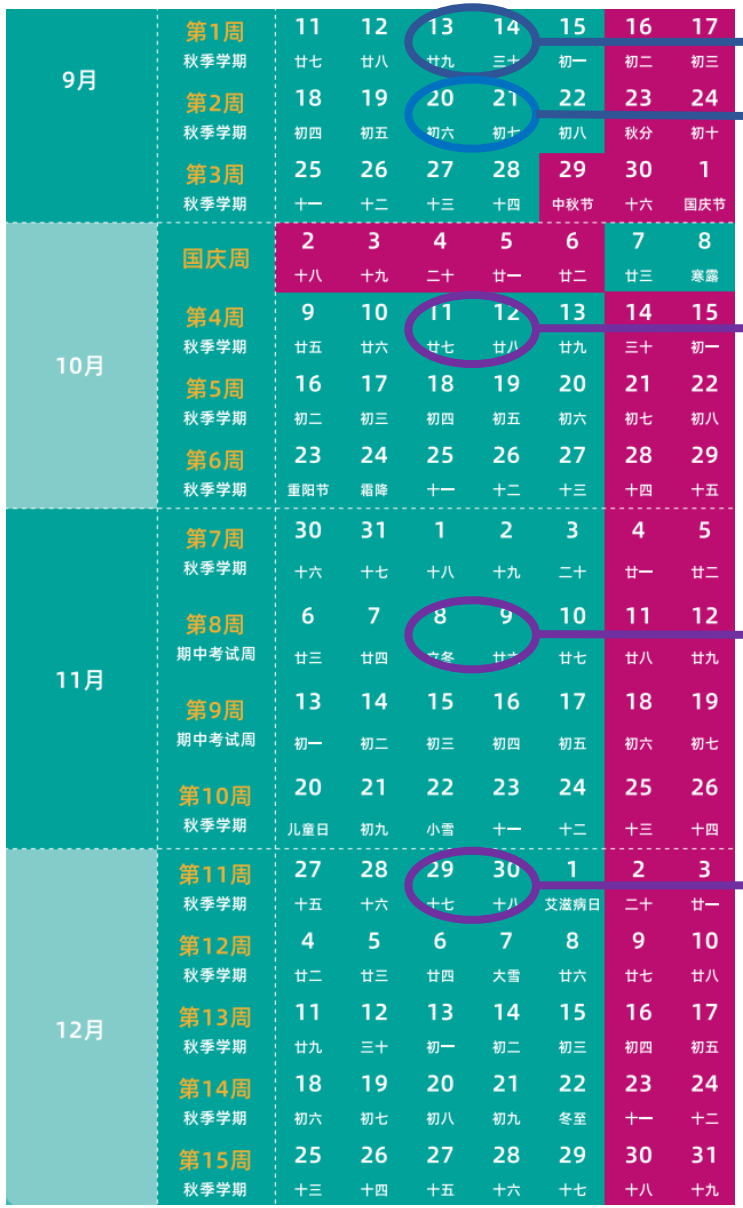
```
module top(
    input btn_p,              // pause button
    input btn_spdup,          // speed-up button
    input btn_spddn,          // speed-down button
    input clk,                // input clk (frequency 100MHz)
    output [7:0] anode,       // anodes for 7-segment
    output [6:0] cathode,     // cathodes for 7-segment
    output dp,                // dot point for 7-segment
    output [7:0] led          // output current addr by led
    );
```

7. We will give you the initial ".s" file to initialize your ROMs the day before the checkpoint. You should build the assembly code to machine code and initialize your memory. Before that, you can initialize your memory whatever you like in simulation.

A sample block diagram



Appendix, Lab Schedule



9月	第1周 秋季学期	11 廿七	12 廿八	13 廿九	14 三十	15 初一	16 初二	17 初三
	第2周 秋季学期	18 初四	19 初五	20 初六	21 初七	22 初八	23 秋分	24 初十
	第3周 秋季学期	25 十一	26 十二	27 十三	28 十四	29 中秋节	30 十六	1 国庆节
10月	国庆周	2 十八	3 十九	4 二十	5 廿一	6 廿二	7 廿三	8 寒露
	第4周 秋季学期	9 廿五	10 廿六	11 廿七	12 廿八	13 廿九	14 三十	15 初一
	第5周 秋季学期	16 初二	17 初三	18 初四	19 初五	20 初六	21 初七	22 初八
	第6周 秋季学期	23 重阳节	24 霜降	25 十一	26 十二	27 十三	28 十四	29 十五
11月	第7周 秋季学期	30 十六	31 十七	1 十八	2 十九	3 二十	4 廿一	5 廿二
	第8周 期中考试周	6 廿三	7 廿四	8 立冬	9 廿六	10 廿七	11 廿八	12 廿九
	第9周 期中考试周	13 初一	14 初二	15 初三	16 初四	17 初五	18 初六	19 初七
	第10周 秋季学期	20 儿童日	21 初九	22 小雪	23 十一	24 十二	25 十三	26 十四
12月	第11周 秋季学期	27 十五	28 十六	29 十七	30 十八	1 艾滋病日	2 二十	3 廿一
	第12周 秋季学期	4 廿二	5 廿三	6 廿四	7 大雪	8 廿六	9 廿七	10 廿八
	第13周 秋季学期	11 廿九	12 三十	13 初一	14 初二	15 初三	16 初四	17 初五
	第14周 秋季学期	18 初六	19 初七	20 初八	21 初九	22 冬至	23 十一	24 十二
	第15周 秋季学期	25 十三	26 十四	27 十五	28 十六	29 十七	30 十八	31 十九

Vivado installation and quick start. Verilog basic and practice.

Lab1 release

Lab1 checkpoint
Lab2 release

Lab2 checkpoint
Lab3 release

Lab3 checkpoint
Project release

Q & A

Q & A

Q & A

Q & A

Here Now

Again, **no report is needed** in this lab, but we will check the **simulation result** and **FPGA demo** in person, and **ask some questions** to ensure you worked this out by yourself.