Keil Debugger Tutorial

Yifeng Zhu December 17, 2014

Software vs Hardware Debug

There are two methods to debug your program: software debug and hardware debug. By using the software debug, you do not have to have the hardware board to debug a software program. However, the hardware debug requires you to connect the board to the computer.

🔣 Options for Target 'Target 1'		Options for Target 'Target 1'	×				
Device Target Output Listing User C/C++ Asm	Linker Debug Utilities	Device Target Output Listing User C/C++ Asm	Linker Debug Utilities				
Use Simulator Settings	○ Use: ST-Link Debugger Settings	C Use Simulator Settings 🔽 Use: ST-Link Debugger 💌 Se					
Limit Speed to Real-Time	·	Limit Speed to Real-Time					
Load Application at Startup Run to main() Initialization File: Load Application File:	Load Application at Startup Run to main() Initialization File: Edit	Load Application at Startup Run to main() Initialization File: Load Application File:	Load Application at Startup Run to main() Initialization File: Load Application File:				
Restore Debug Session Settings	Restore Debug Session Settings	Restore Debug Session Settings	Restore Debug Session Settings				
✓ Breakpoints Toolbox	Breakpoints Toolbox	Breakpoints 🔽 Toolbox	Breakpoints Toolbox				
✓ Watch Windows & Performance Analyzer	✓ Watch Windows	Watch Windows & Performance Analyzer	Vatch Windows				
Memory Display 🔽 System Viewer	Memory Display Vistem Viewer	Memory Display Visystem Viewer	Memory Display System Viewer				
CPU DLL: Parameter:	Driver DLL: Parameter:	CPU DLL: Parameter:	Driver DLL: Parameter:				
SARMCM3.DLL -REMAP -MPU	SARMCM3.DLL -MPU	SARMCM3.DLL REMAP -MPU	SARMCM3.DLL -MPU				
Dialog DLL: Parameter:	Dialog DLL: Parameter:	Dialog DLL: Parameter:	Dialog DLL: Parameter:				
DCM.DLL PCM3	TCM.DLL pCM3	DCM.DLL pCM3	TCM.DLL PCM3				
	, ,						
ОК Са	ncel Defaults Help	ОК Са	ncel Defaults Help				

Selecting software debug

Selecting hardware debug

Debug Control

Debug

Breakpoint

- You can program the STM32 flash by clicking the LOAD button ¹⁰
- Click the debug button 🤐 to start the debug and click it again to exit the debug. You can use the

breakpoint button 💌 to set a break point in either disassembly or source windows.

- STM32 allows up to six breakpoints during hardware debugging. When a program stops at a breakpoint, the corresponding instruction has not been executed yet.
- If the disassembly window is in focus, the debugger executes assembly instructions step by step. If the source window is focused, the debugger then steps through the source lines instead.

		* 🔘 (• • 🔗 🍓 🛛			
The following table su	ummarizes co	mmonly used	debug contro	ol buttons.		
٩	٠	1	8	{ •}	Ω^{+}	{} -
Start	Set a	Run	Stop	Step In	Step Over	Step Out

• **Run**: Continues the execution from the current position until you click *Stop* or the program is paused by a breakpoint.

Debug

• Step In: Execute one step and enter the function if the current step calls a function.

- Step Over: Execute one step and run the function all at once if the current step calls a function.
- Step Out: Execute until the current function returns.



Memory Window

The memory window is used to view the memory content in real time. By default, the address of data memory (RAM) starts at 0x2000_0000. This is specified in the scatter-loading file (*.sct).

The following assembly program defines and allocates an array of four words. Each word consists of four bytes. When we type the memory address 0x20000000, we can see the content of this array.

```
AREA myData, DATA, READWRITE
ALIGN
array DCD 1, 2, 3, 4
```

The memory content is displayed in bytes by default.

	Memory 1																	ņ	
	Address: 0x2000	0000)														I	ſ	•
	0x20000000:	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00		
	0x20000010:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000020:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000050:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000060:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000070:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
	0x20000080:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
1	0x20000090:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		Ŧ
	Call Stack + L	ocals		Mer	nory	1													

By right click, we can select different display format. For example, we can show the content as unsigned integers.

Memory	1					д 🗵
Address	: 0x2000000					<u> </u>
0x2000	0000: 0000001 00000	0002	00000003	00000004	0000000	0
0x2000	0014: 00000000 00000	0000	00000000	00000000	0000000	0
0x200	Decimal	00	00000000	00000000	0000000	0
0x200	Unsigned		Chan	00000000	0000000	0
0x200	Unsigned	<u> </u>	Char	00000000	0000000	0
0x200	Signed	▶ 🗸	Int	00000000	0000000	0
0x200			Short	00000000	0000000	0
0x200	Ascii		2	00000000	0000000	0
0x200	Float		Long	00000000	0000000	0
0 x 200	Double	00	00000000	00000000	0000000	• •
Call	Add '0x20000000' to	•		t1: 0.0002290)8 sec	L:34 C:1

Save Memory Content to a File

In the debug environment, run the following command in the Command Window:

SAVE <filename> <start address>, <end address>

This allows you to perform data analysis in other software tools, such as Microsoft Excel and Matlab. The output is saved in Intel HEX format.

For example, SAVE memory.dat 0x2000000, 0x20000888

```
Command 

Running with Code Size Limit: 32K

Load "C:\\Users\\zhu\\Dropbox\\ECE271\\Labs\\Lab_01_LED_C\\Lab_01_LED_C\\

*** Restricted Version with 32768 Byte Code Size Limit

*** Currently used: 964 Bytes (2%)

BS \\Project\main.c\101

LA `debug_test

LA `NVIC_ICTR

SAVE memory.dat 0x20000000, 0x20000888
```

Processor Registers

Registers	Ф 🖪
Register	Value
Core	
R0	0x20000068
R1	0x0000000
R2	0x40020400
R3	0x20000268
R4	0x0000000
R5	0x20000004
R6	0x0000000
R7	0x0000000
R8	0x0000000
R9	0x0000000
R10	0x080003C0
R11	0x0000000
R12	0x20000044
R13 (SP)	0x20000668
R14 (LR)	0x0800017F
R15 (PC)	0x08000218
⊡ xPSR	0x21000000
N	0
Z	0
C C	1
V	0
QQ	0
T	1
IT	Disabled
ISR	0
Banked	
MSP	0x20000668
PSP	0x0000000
System	
BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x00
□ □····· Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	4111
Sec	0.00051388
🖻 Project 🚟 Regist	ers

Core Registers:

- Program counter (PC) r15 holds the memory address (location in memory) of the next instruction to be fetched from the instruction memory.
- Stack point (SP) r13 holds a memory address that points to the top of the stack. SP is a shadow of either MSP or PSP.
- xPSR (Special-purpose program status registers) is a combination of the following three processor status registers:
 - Application PSR
 - Interrupt PSR
 - Execution PSR

Ν	Negative or less than flag (1 = result negative)
Ζ	Zero flag (1 = result 0)
С	Carry or borrow flag (1 = Carry true
	or borrow false)
V	Overflow flag (1 = overflow)
Q	Q Sticky saturation flag
Т	Thumb state bit
IT	If-Then bits
ISR	ISR Number (6 bits)

System:

- Base priority mask register (BASEPRI) defines the minimum priority for exception processing.
- Priority mask register (PRIMASK) is used to disable all interrupts excluding hard faults and non-maskable interrupts (NMI). If an interrupt is masked, this interrupt is ignored (i.e. disabled) by the processor.
- Control register (CONTROL) sets the choice of main stack or process stack, and the choice of privileged or unprivileged mode.
- Fault mask register (FAULTMASK) is used to disable all interrupts excluding non-maskable interrupts (NMI).

Peripheral Registers

From the menu: **Peripherals** \rightarrow **System Viewer**, we can view and update the control and data registers of all available peripherals. The following figures show all registers for GPIO Port A, such as mode register (MODER), output type register (OTYPER), Output Speed Register (OSPEEDER), Input Data Register (IDR) and Output Data Register (ODR). This provides great conveniences for debugging.



[Bits 31..0] RW (@ 0x40020014) GPIC port output data register

Logic Analyzer

The Logic Analyzer can display the history values (trace) of static or global variables over runtime. Local variables cannot be displayed. The value stored in a register cannot be analyzed.

In the following program, we use logic analyzer to monitor a variable "output" defined in the data area.



In the Logic Analyzer, you can click "Setup" and add a variable "(signed int)output" to observe. Make sure to adjust the data display range to show the curve. Logic analyzer can only monitor global variable. Thus you need to add "**EXPORT output**" to make the output as a global variable.

Setup Logic Analyzer	X
Current Logic Analyzer Signals:	
(signed int)output	
<	4
Signal Display	Display Range
Display Type: Analog 💌	Max: 4096
Color:	Min: 0
Hexadecimal Display	
Display Formula (Signal & Mask) >	> Shift
And Mask: OxFFFFFFFF	Shift Right: 0
Export / Import	
Export Signal Definitions	Import Signal Definitions
	Delete actual Signals
Kill All	Close Help

The following is an example output of a sine wave function.

