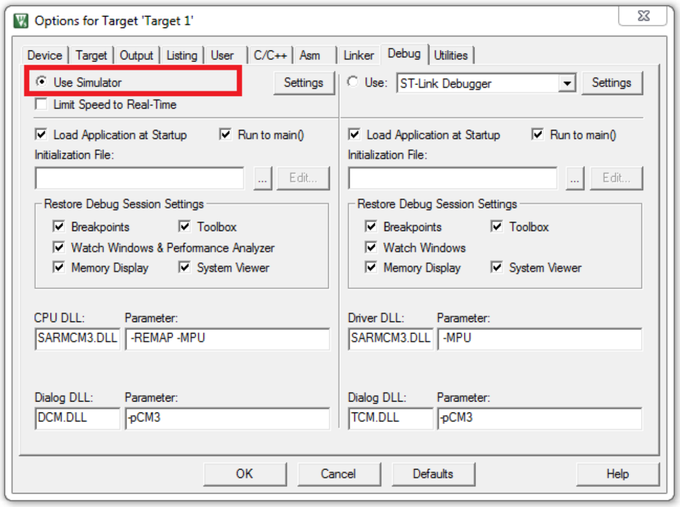
Keil Debugger Tutorial

Yifeng Zhu

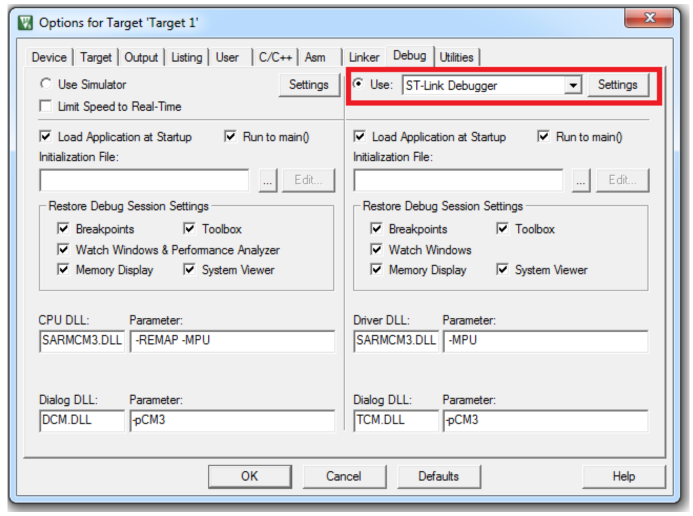
December 17, 2014

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.

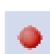


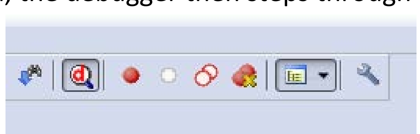
Selecting **software** debug



Selecting **hardware** debug

Debug Control

- You can program the STM32 flash by clicking the LOAD button .
- 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.

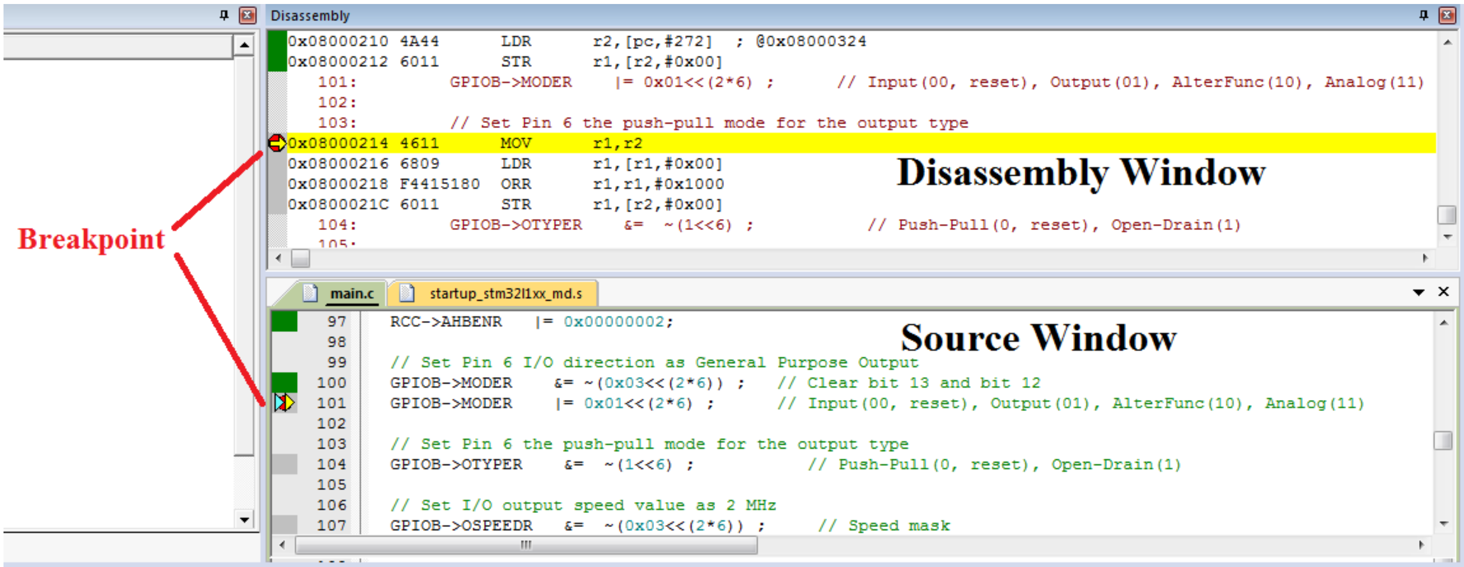


The following table summarizes commonly used debug control buttons.

						
Start Debug	Set a Breakpoint	Run	Stop Debug	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.
- **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.



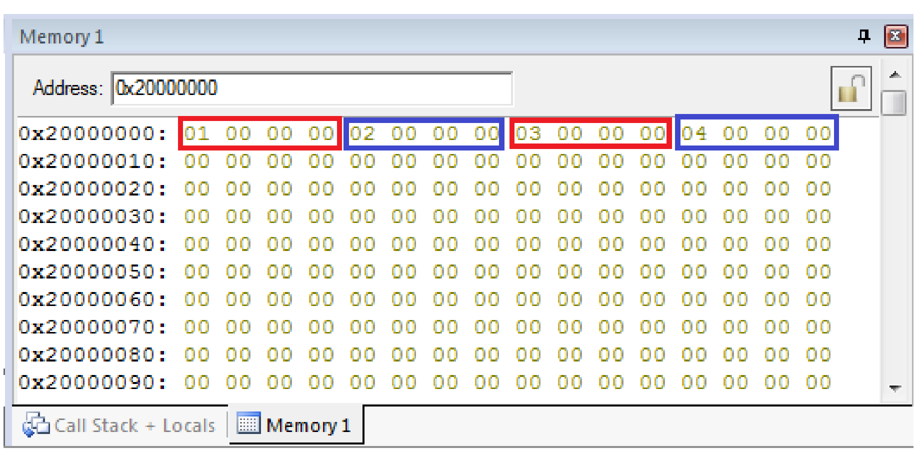
Memory Window

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).

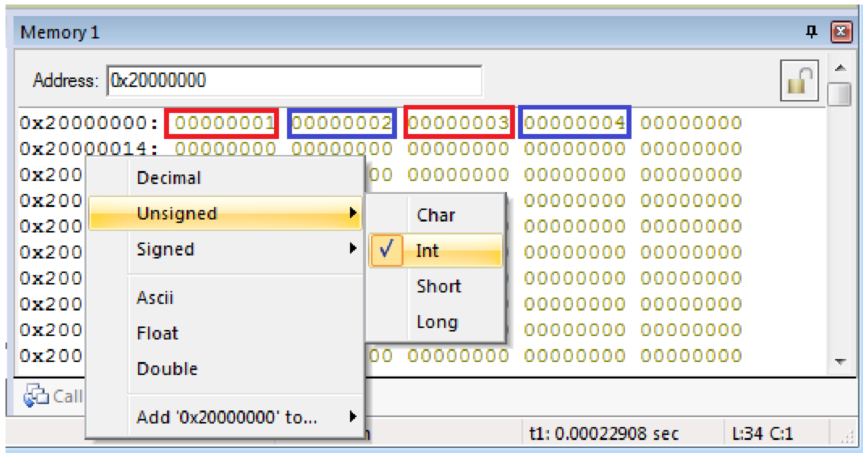
The following assembly program defines and allocates an array of four words. Each word consists of four bytes. When we type the memory address 0x20000000, we can see the content of this array.

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

The memory content is displayed in bytes by default.



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



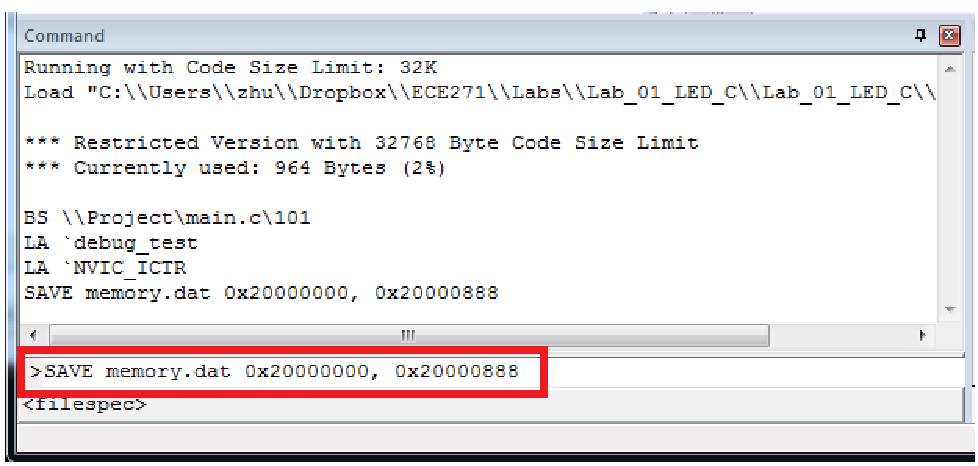
Save Memory Content to a File

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

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

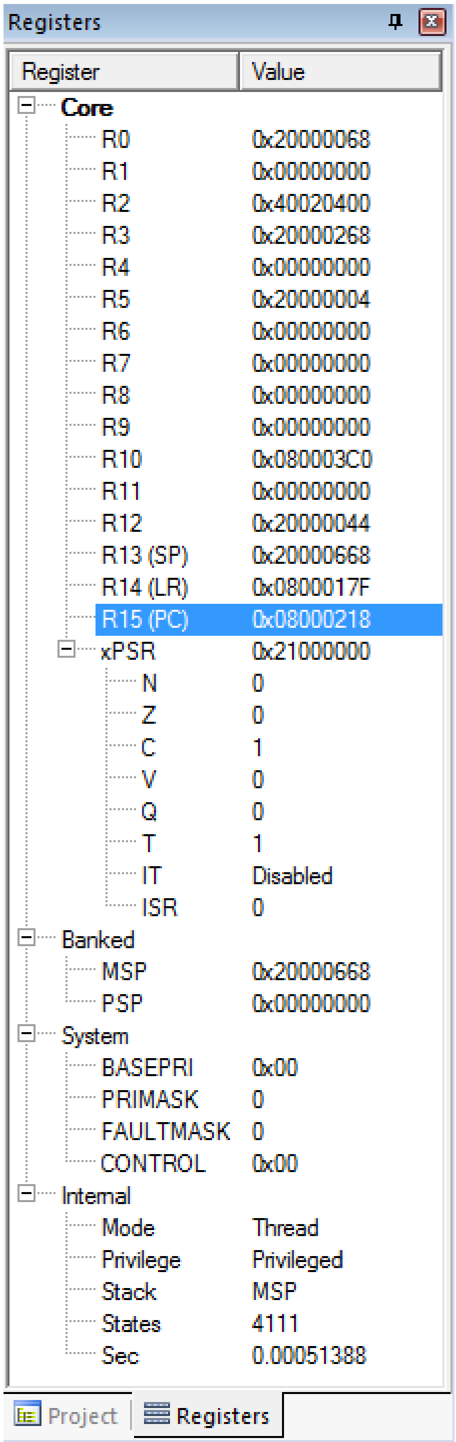
This allows you to perform data analysis in other software tools, such as Microsoft Excel and Matlab. The output is saved in Intel HEX format.

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



 ISBN 0982692625

Processor Registers



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

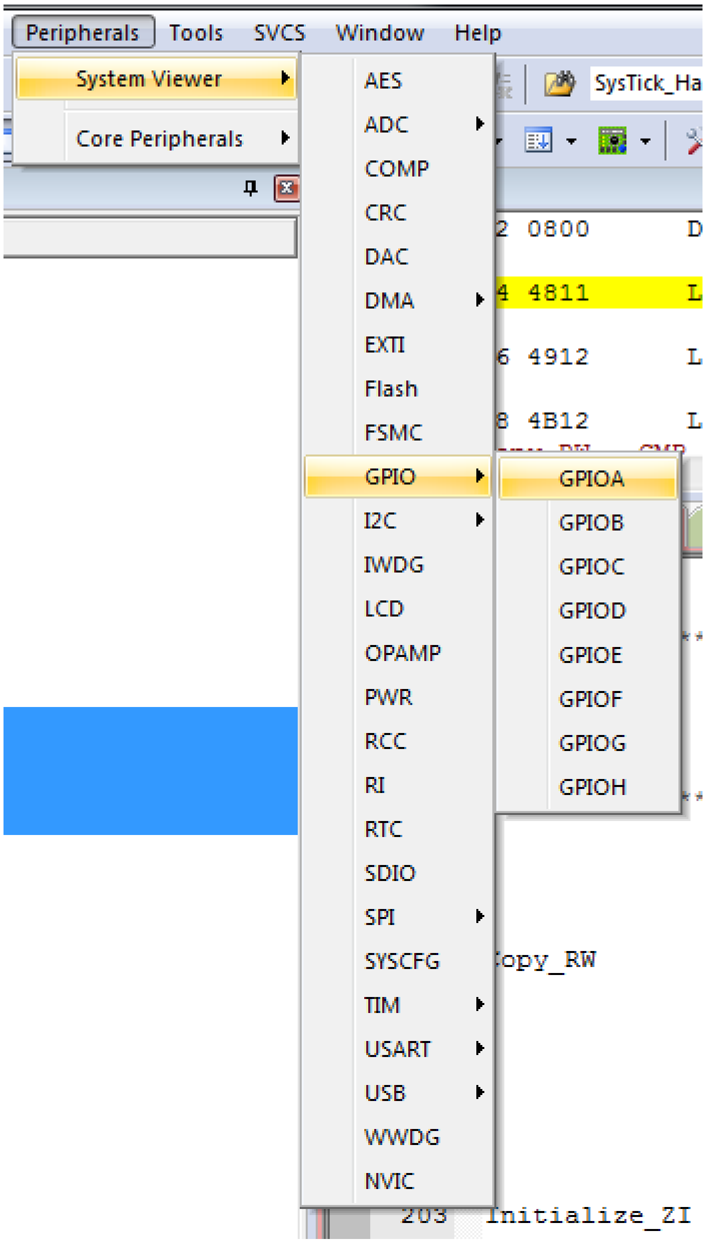
N	Negative or less than flag (1 = result negative)
Z	Zero flag (1 = result 0)
C	Carry or borrow flag (1 = Carry true or borrow false)
V	Overflow flag (1 = overflow)
Q	Q Sticky saturation flag
T	Thumb state bit
IT	If-Then bits
ISR	ISR Number (6 bits)

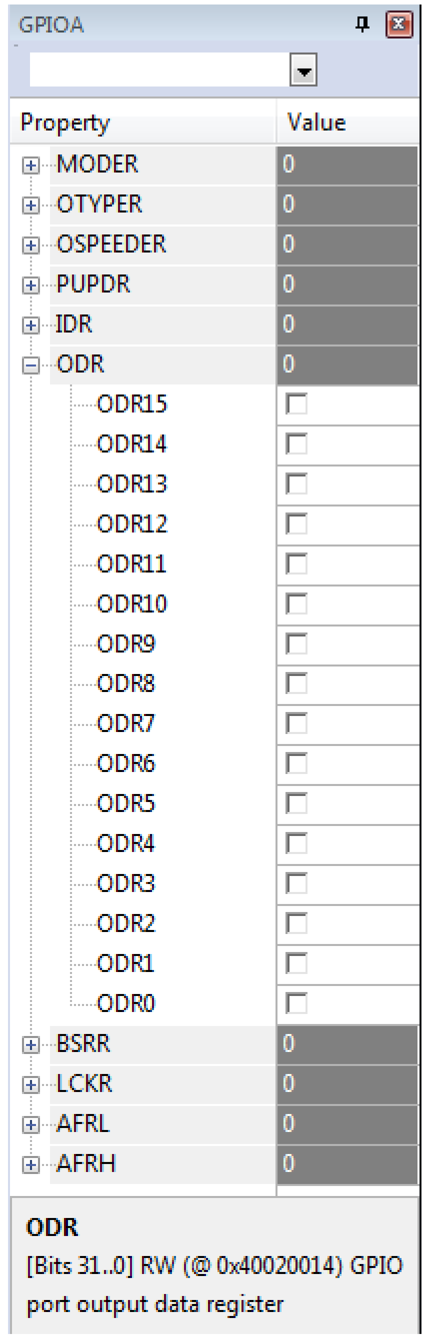
System:

- 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 ⟶ 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.





Logic Analyzer

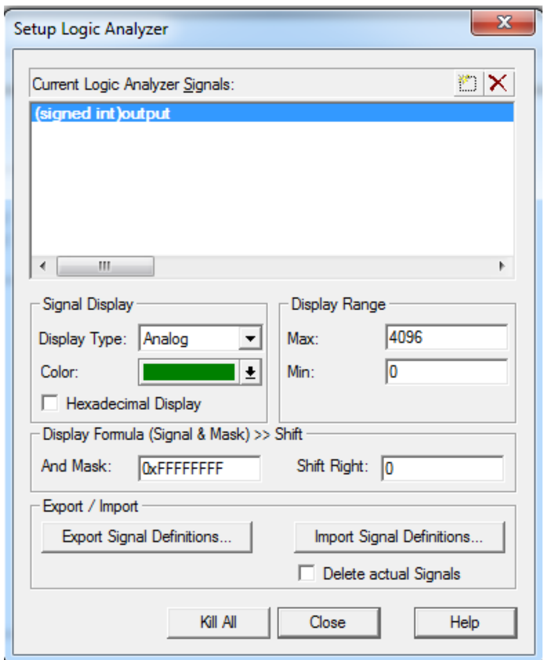
The Logic Analyzer can display the history values (trace) of static or global variables over runtime. Local variables cannot be displayed. The value stored in a register cannot be analyzed.

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

```
        EXPORT output

        AREA myData, DATA
        ALIGN
output  DCD     0x000
```

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.



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