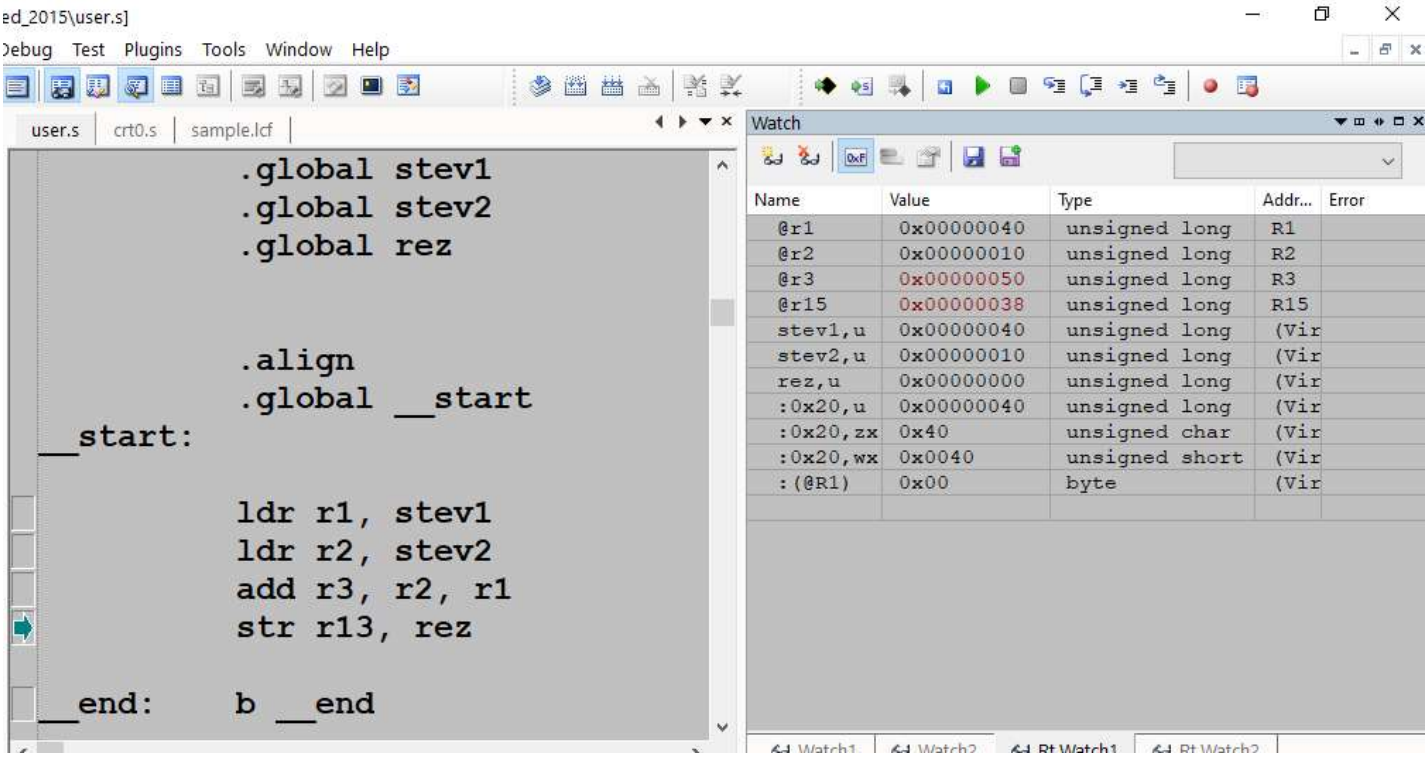
Winidea triki

sobota, 17. oktober 2020 10:56



Watch Expressions

Expression Conventions

A watch expression can be any legal **C expression**, with exception of the ternary '?' operator. Here are some examples of legal expressions:

```
c                variable 'c'
a[3]                third element of array 'a'
a[c+3]*2             can you figure it out?
```

The value of the watch can be further formatted using the **ANSI C printf syntax**:

- ifloat,"%4.2f" ifloat variable printed 4 characters wide, with two decimal digits

⚠	*The type of the expression must match the format string type, otherwise an* Incompatible format *type result is displayed as value.*

The expression syntax has been extended to allow usage of registers and absolute memory locations in expressions:

1. use the '@' prefix to specify a register by the **register name**:

   `@R1` (value of R1 register)

2. use the '@' prefix to specify **full path to a register / bit field**. This allows display of SFRs with identical names belonging to different CPU modules:

   `@"PCU Power Control Unit"\PCU_PCONF0\STBY0`
   Use quotes to wrap the module names separated by spaces.
   Alternatively to typing full SFR path, either SFR or a specific bit field can be dragged from the SFR view to the watch window.
   Same syntax can be used to access SFR registers via isystem.connect.

3. use the ':' prefix to access **absolute memory**. If the CPU has more than one memory area, use memory area specifier before the colon:

   ```
   :0x1000       (byte on address 1000h)
   XDATA:0x30                  (byte on XDATA 30h)
   ```

4. use the '`' prefix to access **I/O module pins**.

	*Character is a **grave accent** (ASCII code 96), not an apostrophe:*

`DIN.DIN0 (value on DIN.DIN0 pin)

I/O module pins can be renamed in the Hardware / Options / I/O tab and in such case the new user-defined names must be used (e.g. DIN.myPinName).

Z naslova <https://www.isystem.com/downloads/winIDEA/help/watchwindow.html>

Watch expression type modifiers

winIDEA will display high level variable values formatted according to their type. If you wish to override default formatting or to enforce a type to a typeless expression, you can use type modifiers.
A type modifier is a comma followed by a set of characters after the expression:
`<expression>,<type modifier>`
Example:
`:0x1000,I` (7281)
Same can be also selected via **Format** command in context menu.
The following type modifiers are available:

- c - force 8 bit signed integer type
- z - force 8 bit unsigned integer type
- i - force 16 bit signed integer type
- w - force 16 bit unsigned integer type
- l - force 32 bit signed integer type
- u - force 32 bit unsigned integer type
- ll - force 64 bit signed integer type
- ull - force 64 bit unsigned integer type
- f - force 32 bit float type
- e - force 64 bit float type
- s - force string interpretation
- a - force array interpretation
- d - decimal formatting
- x or h - hexadecimal formatting
- b - binary formatting
- m - hexadecimal dump
- UTF16 - UTF-16 formatting
- r - display struct member names
- <number> - repeat <number> times

Example 1:
`:0x1000,I -> 7281`
displays locations 0x1000-0x1001 as a signed 16-bit integer.
`:0x1000,F2 -> (4.01,3.14)`
displays locations 0x1000-0x1007 as an array of two 32-bit floats.
`structure,rx -> (a:0x3,y:0x55)`
displays variable 'structure', showing member names and their values formatted hexadecimal.
`largearray,a500`
displays the array 'largearray', starting from the 500th element in the array. This enables watching arrays greater than 256 bytes.

Example 2:
When trying to monitor stack pointer content, it is very convenient to monitor it in the watch window.
To monitor SP content, use following syntax in the watch pane:
`:(@SP-8),8m` displays 8 bytes down from current SP
`:(@SP),6m` displays 6 bytes up from current SP

Array offset and number of elements displayed

Displaying elements of large arrays would slow down the debug experience, so it is advised to limit the part of the array you wish to display by using the appropriate type modifiers.
Format:
`<watch expression>,a[<first element][.<number of elements>]`
Example:

```
sz,a3              // display array elements starting from sz[3]
sz,a3.2           // display 2 array elements, starting at sz[3]
sz,a.2                  // display first 2 array elements
```

Same can be achieved by selecting **Set array offset** format option in the context menu.

Display of char type arrays

Arrays of type char can be displayed in watch window as zero terminated strings. Enable *Debug / Debug Options / Symbols / Display char arrays as zero terminated strings* option to do so.
If the option is not checked, the string is displayed as an array of 8-bit characters. Individual elements are displayed according to **Character display configuration** in the same dialog.
The global setting can be explicitly overridden for individual watches with watch modifiers as described in the chapter above.

Setting write access hardware breakpoint

Variables configured in the Watch window can be used to directly set a hardware **write access hardware breakpoint**.
When a single variable whose address is linear in memory space (i.e. not in a register, or using a register offset), **Set Write Breakpoint** command in the context menu configures the hardware access breakpoint logic (if available on the current platform).

Creating initialization scripts

Watches Window can create a Python initialization script. All selected watches (which can be modified) are exported to a **Python file** along with their current values:

1. Select the watch expressions.
2. Right click and select **Create Initialization Script** command.
3. Specify the file name and save the Python script. The script is using standard isystem.connect Python syntax and can be executed as such.
4. To initialize the variables to their saved state, select in the main toolbar *Tools / External Scripts*.
5. A menu with all available Python scripts will be opened. Select the required script the variable contents will be restored.

⚠	*If regular watch pane is used, monitor access will be used to perform the modification. If real-time watch pane is used, real-time access will be used. This setting can be manually overridden by editing the Python file.*

Creating initialization script with char variables

When generating an initialization script from the watch window variable values will be used as displayed in the watch window. This may be problematic for the char variables, as most commonly they are displayed in ASCII (integer) format. This format will not work in the initialization script. In order to change the display format of the char variables go to *Debug / Debug options / Symbols / Character display and choose Integer*.

Normal time vs. Real-time Watch

When the CPU is running, the Real-time Watch attempts to show every change of a variable's content. A Normal Watch marks every change since the CPU was last stopped. Therefore changes are displayed permanently (until the next CPU stops) whilst changes in real-time watch are displayed only until they change again.

The Real-Time Watch feature attempts to update all listed variable as often as possible.
Several factors limit the speed of the updates:

- The more variables that are selected, the slower the updates will occur.
- The feature is highly dependent on the target's CPUs debug hardware and the speed of the debug interface.

By default real-time update is 0.2 s and it can be changed by clicking *Debug / Debug Options / Update*.

⚠	*This feature may impact the real-time performance of your application and the debug interface may be stealing read/write accesses from the microcontroller's internal bus system.*

More information about Memory Access and Real-time update in Debug Options.

Z naslova <https://www.isystem.com/downloads/winIDEA/help/watchwindow.html>