Data-value watchpoint → DebugMon on STM32L4 (Cortex-M4)

Goal: generate a **DebugMonitor exception** (not a debugger halt) whenever the value **0x1917** is written to **address 0x2000B438** on an STM32L4xx.

On Cortex-M4 this can be done in pure firmware using the **DWT** (Data Watchpoint & Trace) unit in "data value match" mode plus the **DebugMonitor** exception.[1] [2] [3]

Below is a complete, minimal setup assuming:

- Cortex-M4 core (STM32L4xx) with DWT present.
- The write is a **16-bit halfword** write of `0x1917` to `0x2000B438`.
- No halting debugger attached when you want DebugMon to fire (see caveats at the end). [2] [4]

1. Enable trace + DebugMonitor exception

This must be done once at startup:

```
#include "stm32l4xx.h"   // or your device header, pulls in CMSIS core_cm4.h

static void DebugMon_Init(void)
{
    /* Enable trace/DWT and DebugMonitor exceptions */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk   /* enable DWT/ITM/etc */ |
                        CoreDebug_DEMCR_MON_EN_Msk;  /* route debug events to DebugMon */

    /* Set DebugMonitor priority and enable it */
    NVIC_SetPriority(DebugMonitor_IRQn, 0);          /* choose a suitable priority */
    NVIC_EnableIRQ(DebugMonitor_IRQn);
}
```

- `TRCENA` is required for DWT to work at all.[3] [1]
- `MON_EN` allows DWT watchpoint events to raise **DebugMon** instead of just being handled as halting debug events when no debugger is attached.[5] [4] [2]

Call `DebugMon_Init()` early in your startup (before enabling the watchpoint).

2. Configure DWT comparators for “address AND value” match

ARM DWT can compare either **addresses** or **data values**; for a *value at a specific address* you typically use:

- **Comparator 0**: address match (`0x2000B438`)
- **Comparator 1**: data value match (`0x1917`) linked to comparator 0 [6] [1]

On most Cortex-M4s (including STM32L4), **data value matching is only supported on comparator 1.** [1] [3]

2.1. Helper: configure the watchpoint

```
static void Watchpoint_1917_At_2000B438_Init(void)
{
    /* Make sure DWT is accessible */
    if ((DWT->CTRL & DWT_CTRL_NUMCOMP_Msk) == 0) {
        /* No comparators implemented - give up or assert here */
        return;
    }

    /* Optional: clear previous comparator configuration */
    DWT->FUNCTION0 = 0;
    DWT->FUNCTION1 = 0;

    /* -------- Comparator 0: address 0x2000B438 -------- */
    DWT->COMP0 = 0x2000B438UL;
    DWT->MASK0 = 0;         /* no masking → watch exactly one word (aligned) */
    /* Leave FUNCTION0 = 0: it's used only as an address filter for comp1 */

    /* -------- Comparator 1: data value 0x1917 (halfword) -------- */

    /* For data-value watchpoints the DWT compares a 32-bit COMPx value.
       For halfword size the 16-bit pattern must be duplicated into both halfwords.[^1_4]
    uint32_t val16 = 0x1917U;
    DWT->COMP1 = val16 | (val16 << 16);  /* 0x19171917 */

    DWT->MASK1 = 0;   /* no masking of the value itself */

    /* FUNCTION1 fields (see CMSIS core_cm4.h / ARM docs)[^1_30]:
       - FUNCTION[3:0] = 0x6 → generate debug event on *write* data access
       - DATAVMATCH    = 1   → enable data value comparison
       - DATAVSIZE     = 01b → halfword (2 bytes)
       - DATAVADDR0/1  = 0   → use comparator 0 as address filter (default when 0)
    */
    DWT->FUNCTION1 =
          DWT_FUNCTION_DATAVMATCH_Msk          /* enable data value match    */
        | (1U << DWT_FUNCTION_DATAVSIZE_Pos)   /* size = 2 bytes (halfword)  */
        | 0x6U;                                /* write-data watchpoint       */
}
```

Notes:

- `DWT_FUNCTION_DATAVSIZE_Pos`, `DWT_FUNCTION_DATAVMATCH_Msk`, etc. are defined by CMSIS (`core_cm4.h`).[3]
- With `DATAVMATCH=1` and `FUNCTION=0x6`, comparator 1 emits a **watchpoint debug event** whenever a write of the given value (`0x1917` halfword) occurs *at addresses selected by comparator 0*.[6] [1]
- Because `DATAVADDR0`/`DATAVADDR1` fields default to 0, they implicitly point to **comparator 0** when left at reset, so you do not need to set them explicitly.[3] [6]

If instead your code performs a **32-bit** store of `0x00001917`, change two things:

- Use word size in `DATAVSIZE`:

  ```
  DWT->COMP1 = 0x00001917UL;
  DWT->FUNCTION1 =
        DWT_FUNCTION_DATAVMATCH_Msk
      | (2U << DWT_FUNCTION_DATAVSIZE_Pos)  /* word (4 bytes) */
      | 0x6U;
  ```

- And ensure that the store is indeed a 32-bit write, otherwise the watchpoint will not trigger.

2.2. Putting init together

Call both initializers once:

```
int main(void)
{
    /* … system clock, HAL init, etc … */

    DebugMon_Init();
    Watchpoint_1917_At_2000B438_Init();

    /* … rest of application … */
}
```

3. Implementing `DebugMon_Handler`

Provide a handler in your firmware. A simple example:

```
void DebugMon_Handler(void)
{
    /* Optional: check that the event really came from DWT */
    uint32_t dfsr = SCB->DFSR;
    if (dfsr & SCB_DFSR_DWTTRAP_Msk) {

        /* Clear the sticky DWTTRAP flag */
        SCB->DFSR = SCB_DFSR_DWTTRAP_Msk;

        /* Check which comparator matched (MATCHED bit) */
        if (DWT->FUNCTION1 & DWT_FUNCTION_MATCHED_Msk) {
            /* Clear MATCHED by reading FUNCTION1 then writing it back with bit 24 cleare
```

```
            uint32_t f1 = DWT->FUNCTION1;
            DWT->FUNCTION1 = f1 & ~DWT_FUNCTION_MATCHED_Msk;

            /* At this point a write of 0x1917 to 0x2000B438 has just occurred.
               You can log state, break into a debug loop, reset, etc. */

            /* Example: place a breakpoint if a debugger *is* attached */
            __BKPT(0);
        }
    }
}
```

In production you might instead:

- Log the fault context (stacked registers, LR/PC, task info).
- Store the event in a ring buffer (address, value, PC) for later retrieval.[7] [2]
- Reset the system or attempt local recovery.

4. Important caveats

1. **Debugger vs DebugMonitor are mutually exclusive in practice**
   When a standard SWD/JTAG debugger is attached in **halting debug** mode, most debug servers configure the core so that watchpoint events **halt the CPU** under the debugger rather than raise `DebugMon`. In this case your `DebugMon_Handler` will not run, or the core may halt before it does.[4] [2]
   - To observe `DebugMon_Handler`, run **without a halting debugger** attached, or use a debug configuration that does not enable halting debug (rare).
   - Some IDEs make this combination tricky; this is an architectural limitation, not STM32-specific.[2] [4]
2. **Access size must match**
   The `DATAVSIZE` field must match the actual bus access size:
   - 0 → byte
   - 1 → halfword (2 bytes)
   - 2 → word (4 bytes)[3]

   If your code sometimes writes `uint8_t`, sometimes `uint16_t`, you need to pick the one relevant to the corruption you are tracking (or set up different watchpoints).
3. **Address alignment**
   - With `MASK0 = 0`, comparator 0 matches exactly the aligned word at `0x2000B438`.[1] [3]
   - If you need a range (e.g. an array), increase `MASK0` to ignore low address bits (see DWT docs for mapping of `MASK` to region size).[1] [3]
4. **Comparator availability**
   - Use `DWT->CTRL & DWT_CTRL_NUMCOMP_Msk` to ensure at least 2 comparators are implemented.

- On STM32L4xx, four comparators are typically present, with **data value matching only on comparator 1.** [8] [1]

Minimal recipe

1. Enable `TRCENA` and `MON_EN` in `CoreDebug->DEMCR`.
2. Enable `DebugMonitor_IRQn` and set its priority.
3. Use **DWT comparator 0** for address `0x2000B438` (COMP0/MASK0).
4. Use **DWT comparator 1** in **data value match** mode for `0x1917` with `DATAVSIZE` set appropriately and `FUNCTION=0x6`.
5. Implement `DebugMon_Handler` to handle and clear the event.

With this in place, any write of `0x1917` of the selected size to `0x2000B438` (while no halting debugger is attached) will raise the **DebugMonitor** exception instead of just silently corrupting memory. [4] [8] [2] [1] [3]

⁂

Data-value watchpoint → DebugMon on STM32L4 (Cortex-M4) – 32-bit word write

Goal: generate a **DebugMonitor exception** whenever the **32-bit word value 0x00001917** is written to **address 0x2000B438** on an STM32L4xx.

The setup is identical to the halfword case, except for two register fields in comparator 1:

- `DATAVSIZE` must be set to `2` (binary `10`) to match a **word** (4 bytes). [53] [54]
- `COMP1` holds the full **32-bit value** `0x00001917` (no duplication needed). [55] [53]

Below is the complete, minimal firmware configuration.

1. Enable trace + DebugMonitor exception

Same as before—call once at startup:

```
#include "stm32l4xx.h"   // or your device header, pulls in CMSIS core_cm4.h

static void DebugMon_Init(void)
{
    /* Enable trace/DWT and DebugMonitor exceptions */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk   /* enable DWT/ITM/etc */ |
                        CoreDebug_DEMCR_MON_EN_Msk;  /* route debug events to DebugMon */

    /* Set DebugMonitor priority and enable it */
    NVIC_SetPriority(DebugMonitor_IRQn, 0);          /* choose a suitable priority */
    NVIC_EnableIRQ(DebugMonitor_IRQn);
}
```

- `TRCENA` is required for DWT to work at all.[54] [53]
- `MON_EN` allows DWT watchpoint events to raise **DebugMon** instead of just being handled as halting debug events when no debugger is attached.[56] [57] [58]

Call `DebugMon_Init()` early in your startup (before enabling the watchpoint).

2. Configure DWT comparators for “address AND 32-bit value” match

ARM DWT can compare either **addresses** or **data values**; for a *value at a specific address* you typically use:

- **Comparator 0**: address match (`0x2000B438`)
- **Comparator 1**: data value match (`0x00001917`) linked to comparator 0[55] [53]

On most Cortex-M4s (including STM32L4), **data value matching is only supported on comparator 1**.[53] [54]

2.1. Helper: configure the watchpoint for 32-bit word writes

```
static void Watchpoint_1917_At_2000B438_Init(void)
{
    /* Make sure DWT is accessible */
    if ((DWT->CTRL & DWT_CTRL_NUMCOMP_Msk) == 0) {
        /* No comparators implemented – give up or assert here */
        return;
    }

    /* Optional: clear previous comparator configuration */
    DWT->FUNCTION0 = 0;
    DWT->FUNCTION1 = 0;

    /* -------- Comparator 0: address 0x2000B438 -------- */
    DWT->COMP0 = 0x2000B438UL;
    DWT->MASK0 = 0;         /* no masking → watch exactly one word (aligned) */
    /* Leave FUNCTION0 = 0: it is used only as an address filter for comp1 */

    /* -------- Comparator 1: data value 0x00001917 (32-bit word) -------- */

    /* For 32-bit word writes, store the full 32-bit value in COMP1. */
    DWT->COMP1 = 0x00001917UL;

    DWT->MASK1 = 0;   /* no masking of the value itself */

    /* FUNCTION1 fields (see CMSIS core_cm4.h / ARM docs)[^2_30]:
       - FUNCTION[3:0] = 0x6 → generate debug event on *write* data access
       - DATAVMATCH    = 1   → enable data value comparison
       - DATAVSIZE     = 10b → word (4 bytes)
       - DATAVADDR0/1  = 0   → use comparator 0 as address filter (default when 0)
    */
    DWT->FUNCTION1 =
          DWT_FUNCTION_DATAVMATCH_Msk          /* enable data value match    */
        | (2U << DWT_FUNCTION_DATAVSIZE_Pos)   /* size = 4 bytes (word)      */
        | 0x6U;                                /* write-data watchpoint       */
}
```

Notes:

- `DWT_FUNCTION_DATAVSIZE_Pos`, `DWT_FUNCTION_DATAVMATCH_Msk`, etc. are defined by CMSIS (`core_cm4.h`).[54]
- With `DATAVMATCH=1` and `FUNCTION=0x6`, comparator 1 emits a **watchpoint debug event** whenever a write of the given value (`0x00001917` word) occurs *at addresses selected by comparator 0*.[55] [53]
- Because `DATAVADDR0`/`DATAVADDR1` fields default to 0, they implicitly point to **comparator 0** when left at reset, so you do not need to set them explicitly.[54] [55]

If instead your code performs a **16-bit** store of `0x1917`, change two things:

- Use halfword size in `DATAVSIZE`:

  ```
  DWT->COMP1 = 0x19171917UL;  /* duplicate pattern for halfword */
  DWT->FUNCTION1 =
        DWT_FUNCTION_DATAVMATCH_Msk
      | (1U << DWT_FUNCTION_DATAVSIZE_Pos)  /* halfword (2 bytes) */
      | 0x6U;
  ```

- And ensure that the store is indeed a 16-bit write, otherwise the watchpoint will not trigger.

2.2. Putting init together

Call both initializers once:

```
int main(void)
{
    /* … system clock, HAL init, etc … */

    DebugMon_Init();
    Watchpoint_1917_At_2000B438_Init();

    /* … rest of application … */
}
```

3. Implementing `DebugMon_Handler`

Provide a handler in your firmware. A simple example:

```
void DebugMon_Handler(void)
{
    /* Optional: check that the event really came from DWT */
```

```
    uint32_t dfsr = SCB->DFSR;
    if (dfsr & SCB_DFSR_DWTTRAP_Msk) {

        /* Clear the sticky DWTTRAP flag */
        SCB->DFSR = SCB_DFSR_DWTTRAP_Msk;

        /* Check which comparator matched (MATCHED bit) */
        if (DWT->FUNCTION1 & DWT_FUNCTION_MATCHED_Msk) {
            /* Clear MATCHED by reading FUNCTION1 then writing it back with bit 24 cleare
            uint32_t f1 = DWT->FUNCTION1;
            DWT->FUNCTION1 = f1 & ~DWT_FUNCTION_MATCHED_Msk;

            /* At this point a write of 0x00001917 to 0x2000B438 has just occurred.
               You can log state, break into a debug loop, reset, etc. */

            /* Example: place a breakpoint if a debugger *is* attached */
            __BKPT(0);
        }
    }
}
```

In production you might instead:

- Log the fault context (stacked registers, LR/PC, task info).
- Store the event in a ring buffer (address, value, PC) for later retrieval.[57] [59]
- Reset the system or attempt local recovery.

4. Important caveats

1. **Debugger vs DebugMonitor are mutually exclusive in practice**
   When a standard SWD/JTAG debugger is attached in **halting debug** mode, most debug servers configure the core so that watchpoint events **halt the CPU** under the debugger rather than raise `DebugMon`. In this case your `DebugMon_Handler` will not run, or the core may halt before it does.[58] [57]
   - To observe `DebugMon_Handler`, run **without a halting debugger** attached, or use a debug configuration that does not enable halting debug (rare).
   - Some IDEs make this combination tricky; this is an architectural limitation, not STM32-specific.[57] [58]
2. **Access size must match**
   The `DATAVSIZE` field must match the actual bus access size:
   - 0 → byte
   - 1 → halfword (2 bytes)
   - 2 → word (4 bytes)[54]

   If your code sometimes writes `uint8_t`, sometimes `uint32_t`, you need to pick the one relevant to the corruption you are tracking (or set up different watchpoints).
3. **Address alignment**

- With `MASK0 = 0`, comparator 0 matches exactly the aligned word at `0x2000B438`.[53] [54]
- If you need a range (e.g. an array), increase `MASK0` to ignore low address bits (see DWT docs for mapping of `MASK` to region size).[53] [54]

4. **Comparator availability**
   - Use `DWT->CTRL & DWT_CTRL_NUMCOMP_Msk` to ensure at least 2 comparators are implemented.
   - On STM32L4xx, four comparators are typically present, with **data value matching only on comparator 1**.[60] [53]

Minimal recipe for 32-bit word writes

1. Enable `TRCENA` and `MON_EN` in `CoreDebug->DEMCR`.
2. Enable `DebugMonitor_IRQn` and set its priority.
3. Use **DWT comparator 0** for address `0x2000B438` (COMP0/MASK0).
4. Use **DWT comparator 1** in **data value match** mode for `0x00001917` with `DATAVSIZE = 2` and `FUNCTION = 0x6`.
5. Implement `DebugMon_Handler` to handle and clear the event.

With this in place, any **32-bit word** write of `0x00001917` to `0x2000B438` (while no halting debugger is attached) will raise the **DebugMonitor** exception instead of just silently corrupting memory.[58] [60] [57] [53] [54]

⁂

1. https://interrupt.memfault.com/blog/cortex-m-watchpoints
2. https://mcuoneclipse.com/2018/08/12/tutorial-catching-rogue-memory-accesses-with-arm-watchpoint-comparators-and-instruction-trace/
3. https://docs.contiki-ng.org/en/release-v4.2/_api/core__cm3_8h.html
4. https://community.nxp.com/t5/MCUXpresso-IDE/Advanced-instruction-trace/m-p/793522
5. https://cse.buffalo.edu/tech-reports/2024-03.pdf
6. https://www2.lauterbach.com/pdf/debugger_cortexm.pdf
7. https://mcuoneclipse.com/2025/09/25/using-can-fd-for-remote-hardware-debugging-of-cortex-m-devices/
8. https://www.st.com/content/ccc/resource/training/technical/product_training/group0/8a/bf/62/da/24/f0/49/7b/STM32H7-System-Debug_DBG/files/STM32H7-System-Debug_DBG.pdf/_jcr_content/translations/en.STM32H7-System-Debug_DBG.pdf
9. https://interrupt.memfault.com/blog/cortex-m-hardfault-debug
10. https://github.com/zephyrproject-rtos/zephyr/blob/main/arch/arm/core/cortex_m/Kconfig
11. https://mcuoneclipse.com/2017/01/30/cycle-counting-on-arm-cortex-m-with-dwt/
12. https://www.reddit.com/r/embedded/comments/1knyc6k/whats_your_favour_tricks_to_debug_interrupts_for/

13. https://www.infineon.com/dgdl/Infineon-System_Debug_XMC4-TR-v01_01-EN.pdf?fileId=5546d4624bcaebcf014bcb16c9ac004b
14. https://developer.arm.com/documentation/101051/0001/Data-Watchpoint-and-Trace/DWT-debug-access-control
15. https://www.youtube.com/watch?v=4wT9NhlcWP4
16. https://github.com/Marus/cortex-debug/issues/332
17. http://libopencm3.org/docs/latest/samd/html/group__cm__dwt.html
18. https://community.st.com/t5/stm32-mcus/integrating-and-debugging-external-memory-on-stm32-mcus-part-2/ta-p/865238
19. https://www.reddit.com/r/embedded/comments/1gu3zu5/stm32_bitwise_operations_beginner_question/
20. https://cs2461-2020.github.io/lectures/bitwise.pdf
21. https://os.mbed.com/users/yihui/code/mbed-src-nrf51822/raw-rev/700cadd8b708
22. https://stackoverflow.com/questions/40551598/c-bitwise-what-does-1-num-1-and-do
23. https://www.reddit.com/r/embedded/comments/v7gn5g/help_with_bitwise_operations_basics/
24. https://gist.github.com/inkwisit/798874bbda618875012edb5716d0e3f6
25. https://skoopsy.dev/stm32/2025/11/02/9-C-bitwise.html
26. https://baltig.infn.it/carniti/CROSSFrontendBtldr/-/blob/main/Listings/system_LPC17xx.lst
27. https://www.sharetechnote.com/html/C_BitManipulation.html
28. https://files.elektroda.pl/856555,led_blink.html
29. https://libwebsockets.org/git/libwebsockets/commit/minimal-examples/embedded/pico/pico-sspc-binance/README.md?id=2761badd0f190c56d14341ef2531613bea91e263
30. http://gamesmith.tistory.com/m/115
31. https://kolegite.com/gitweb/?p=vmks.git%3Ba%3Dblob%3Bf%3Dstm32f103_firmware%2FDrivers%2FCMSIS%2FInclude%2Fcore_cm3.h%3Bh%3Db0dfbd3d9d43fda35afeaa03d7315f2c075601e2%3Bhb%3DHEAD
32. https://software-dl.ti.com/simplelink/esd/simplelink_lowpower_f3_sdk/7.40.00.64/exports/docs/driverlib/cc27xx/driverlib/core__armv81mml_8h.html
33. https://gitlab.fing.edu.uy/dm3/proyectoRAS/-/blob/6a542fd9297597be20d4f56d45f352ac2b2f8b0a/ras_frdm/mbed/TARGET_K64F/core_sc300.h
34. https://www.hilscher.com/external/netXDriverDocumentation/group___c_m_s_i_s___d_w_t.html
35. https://android.googlesource.com/trusty/lk/common/+/9a66db064a7b0e64d088a0127c53a0090fefd204^!/
36. https://irtos.sourceforge.net/2000/Documentation/doc_developer_en/html/a00238.html
37. https://gitlab.esat.kuleuven.be/pcy.sluys/cw-fw-extra-l4r5z/-/blob/main/imxrt1062/nxp/CMSIS/core_cm7.h
38. https://www.hilscher.com/external/netXDriverDocumentation/core__cm4_8h.html
39. http://docs.ros.org/melodic/api/inertial_sense/html/group__CMSIS__DWT.html
40. https://github.com/Decawave/dwm1001-examples/blob/master/nRF5_SDK_14.2.0/components/toolchain/cmsis/include/core_cm4.h
41. https://gitlab.auto.tuwien.ac.at/auto/zephyr/-/blob/c7297b804a64101f191b1a370d20ce0299f0ce00/arch/arm/core/aarch32/cortex_m/debug.c
42. https://docs.rtems.org/doxygen/main/globals_d.html

43. https://kolegite.com/gitweb/?p=vmks.git%3Ba%3Dblob%3Bf%3Dcore_cm4.h%3Bh%3D308b86813ca7b8cd37610e51d13b735fc6578232
44. https://documentation-service.arm.com/static/5f2286f2f3ce30357bc28b2a?token=
45. https://mcuoneclipse.com/2012/11/24/debugging-hard-faults-on-arm-cortex-m/
46. https://www.diva-portal.org/smash/get/diva2:1669717/FULLTEXT01.pdf
47. https://community.arm.com/support-forums/f/keil-forum/52764/how-do-you-use-dwt-mechanism-to-trigger-interrupt-on-memory-writes
48. https://riscv.org/wp-content/uploads/2024/12/riscv-debug-release.pdf
49. https://forums.freertos.org/t/the-program-execution-is-suddenly-stuck-in-configassert-configassert-pxlink-xblocksize-xblockallocatedbit-0/14267
50. https://devzone.nordicsemi.com/f/nordic-q-a/69203/setting-a-watchpoint-dynamically?pifragment-684=4
51. https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/S32K/18964/1/bist_with_unexpected_reset.txt
52. https://stackoverflow.com/questions/72122224/data-watchpoints-dwt-on-cortex-m4-to-detect-memory-corruption
53. https://interrupt.memfault.com/blog/cortex-m-watchpoints
54. https://docs.contiki-ng.org/en/release-v4.2/_api/core__cm3_8h.html
55. https://www2.lauterbach.com/pdf/debugger_cortexm.pdf
56. https://cse.buffalo.edu/tech-reports/2024-03.pdf
57. https://mcuoneclipse.com/2018/08/12/tutorial-catching-rogue-memory-accesses-with-arm-watchpoint-comparators-and-instruction-trace/
58. https://community.nxp.com/t5/MCUXpresso-IDE/Advanced-instruction-trace/m-p/793522
59. https://mcuoneclipse.com/2025/09/25/using-can-fd-for-remote-hardware-debugging-of-cortex-m-devices/
60. https://www.st.com/content/ccc/resource/training/technical/product_training/group0/8a/bf/62/da/24/f0/49/7b/STM32H7-System-Debug_DBG/files/STM32H7-System-Debug_DBG.pdf/_jcr_content/translations/en.STM32H7-System-Debug_DBG.pdf
61. https://stackoverflow.com/questions/72122224/data-watchpoints-dwt-on-cortex-m4-to-detect-memory-corruption
62. https://github.com/STMicroelectronics/STM32CubeMP1/blob/master/Drivers/CMSIS/Core/Include/core_cm33.h
63. https://www.isystem.com/downloads/winIDEA/help/cortex-dwtcomparator.html
64. https://gitlab.insa-rennes.fr/syns/tp_insa_hwsecurity/-/blob/4dc033aab15cd02a5fc61da784caa0ab9a7d3c6a/firmware/hal/silabs_sdk/platform/CMSIS/Include/core_cm23.h
65. https://community.arm.com/support-forums/f/keil-forum/52764/how-do-you-use-dwt-mechanism-to-trigger-interrupt-on-memory-writes
66. https://www.mimuw.edu.pl/~mm319369/cc2650.html
67. https://gitlab.uwe.ac.uk/sah2-zaheer/esp-lab-4/-/blob/main/Drivers/CMSIS/Core/Include/core_armv81mml.h
68. https://community.st.com/t5/stm32-mcus-products/conditional-data-watchpoint-with-debugger-or-dwt/td-p/219672

69. https://software-dl.ti.com/simplelink/esd/simplelink_cc13xx_cc26xx_sdk/7.41.00.17/exports/docs/driverlib/cc13×2_cc26×2/register_descriptions/CPU_MMAP/CPU_DWT.html
70. https://android.googlesource.com/trusty/lk/common/+/9a66db064a7b0e64d088a0127c53a0090fefd204^!/
71. https://www2.lauterbach.com/pdf/training_cortexm_etm.pdf
72. https://developer.arm.com/documentation/ddi0403/d/Appendices/Debug-ITM-and-DWT-Packet-Protocol/DWT-use-of-Hardware-source-packets/Data-trace-packets-discriminator-IDs-8-23
73. https://stackoverflow.com/questions/39419/how-large-is-a-dword-with-32-and-64-bit-code