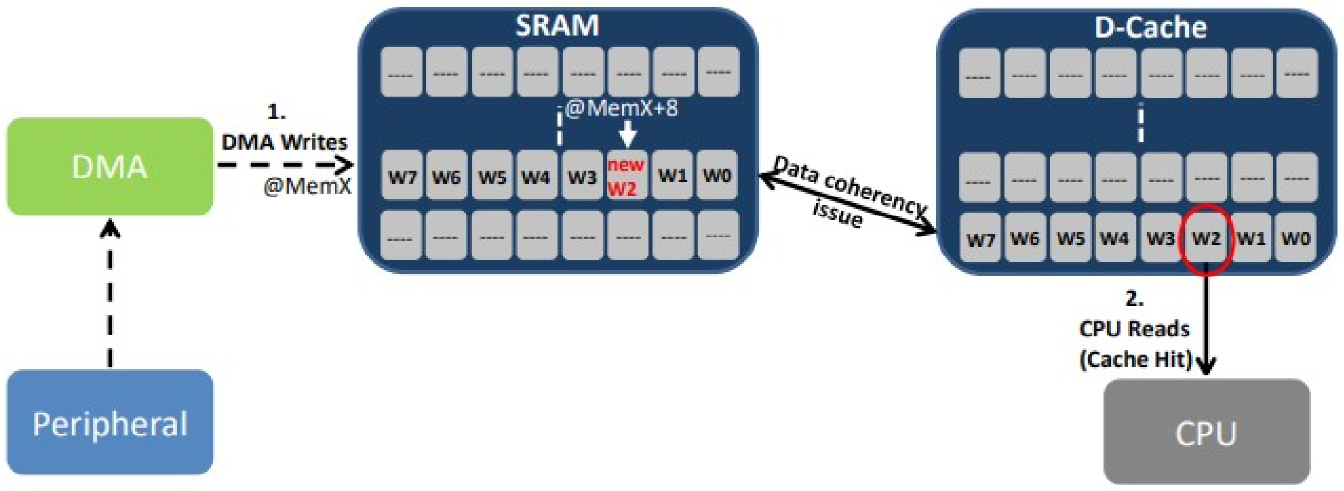
Figure 3-1. Cache Coherency Issue - DMA Writes to SRAM



Where,

1. The DMA reads the data from the peripheral and updates the receive buffer in the SRAM.

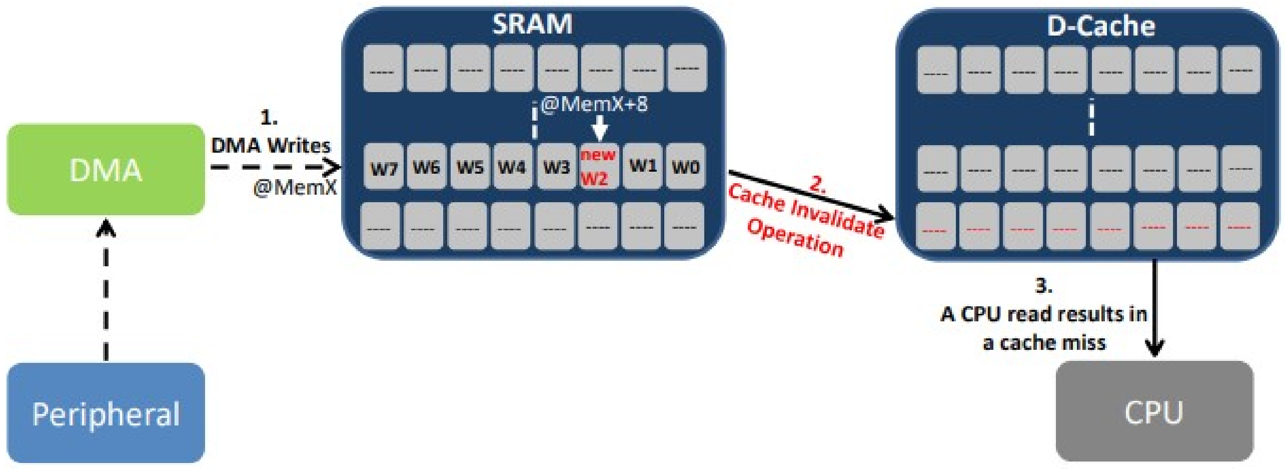
2. When the CPU tries to read the receive buffer, it will read the data present in the cache and not the new data available in the SRAM.

Using cache maintenance API when DMA writes to SRAM

Conditions: The cache policy is WB-RWA. The CPU initially accessed the receive buffer (rx_buffer[]), and cached it in the D-Cache.

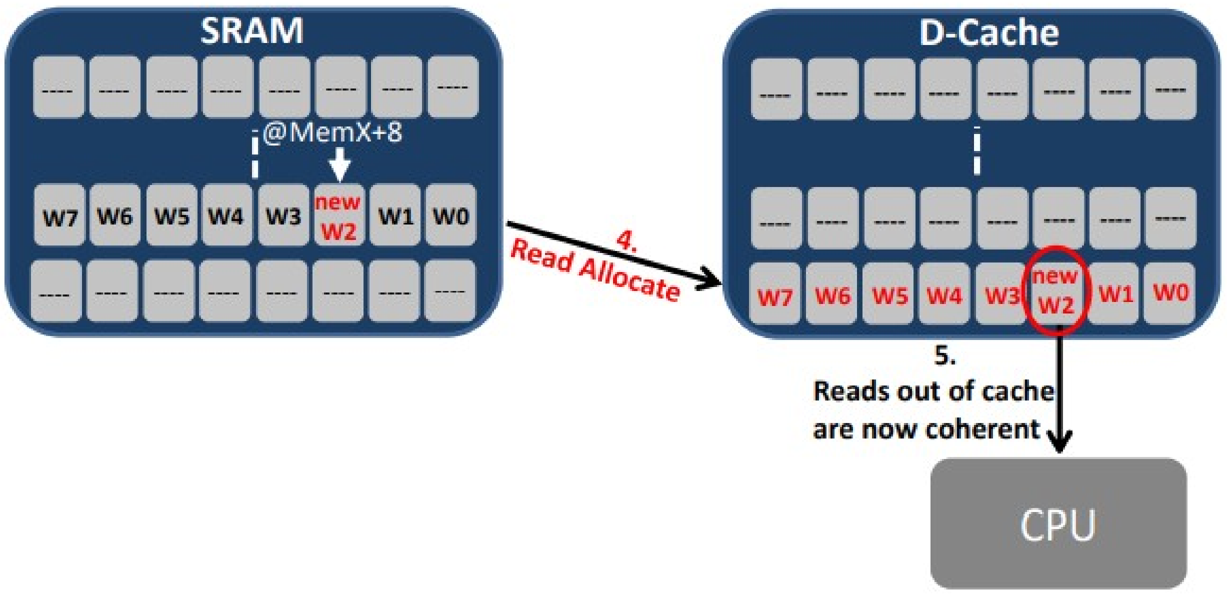
1. DMA writes data to the rx_buffer[].
2. A cache invalidate operation is performed to invalidate the cached rx_buffer[].
3. CPU tries to read the rx_buffer[] and results in a cache miss as rx_buffer[] was invalidated in step 2.

Figure 4-1. Cache Invalidate Operation After DMA Writes to SRAM


SRAM
D-Cache
DMA
Peripheral
CPU
1.
DMA Writes
@MemX
@MemX+8
W7 W6 W5 W4 W3 new W2 W1 W0
2.
Cache Invalidate Operation
3.
A CPU read results in a cache miss

4. Due to the read-allocate policy, a cache line is allocated and copies data from the rx_buffer[] in the SRAM to the allocated cache line.
5. The CPU reads from the cache will then be coherent.

Figure 4-2. After a Cache Invalidate Operation, Reads Out of D-Cache by CPU are Coherent

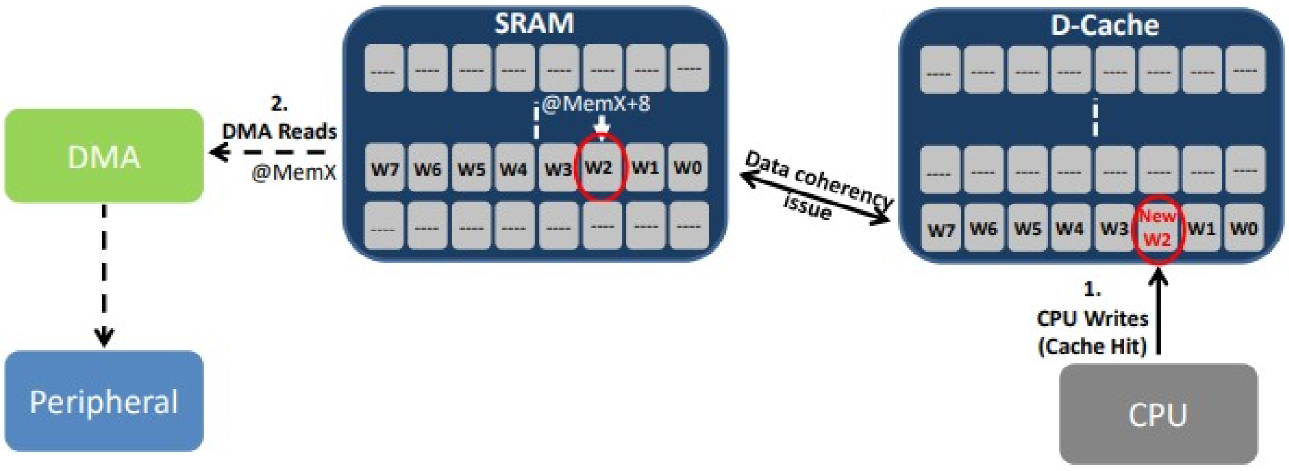

SRAM
D-Cache
@MemX+8
W7 W6 W5 W4 W3 new W2 W1 W0
4.
Read Allocate
W7 W6 W5 W4 W3 new W2 W1 W0
5.
Reads out of cache are now coherent
CPU

```
void XDMAC_Handler(void)
{
    uint32_t dma_status;

    dma_status = xdmac_channel_get_interrupt_status(XDMAC, XDMA_CH_RX);

    if (dma_status & XDMAC_CIS_BIS)
    {
        rx_xfer_done = true;
        SCB_InvalidateDCache_by_Addr((uint32_t*)rx_buffer, DMA_TRANSFER_SIZE);
    }
}
int main (void)
{
    ……
    /* Enabling the D-Cache */
    SCB_EnableDCache();
    /* Setup and trigger a DMA transfer */
    ……
    while (false == rx_xfer_done);
    /* Access to the rx_buffer[] is coherent now */
}
```

Figure 3-2. Cache coherency Issue - DMA Reads from SRAM



Where,

1. The CPU updates the data to be transmitted in a transmit buffer as the cache policy is set to WB-RWA, only the cache is updated and not the main memory.

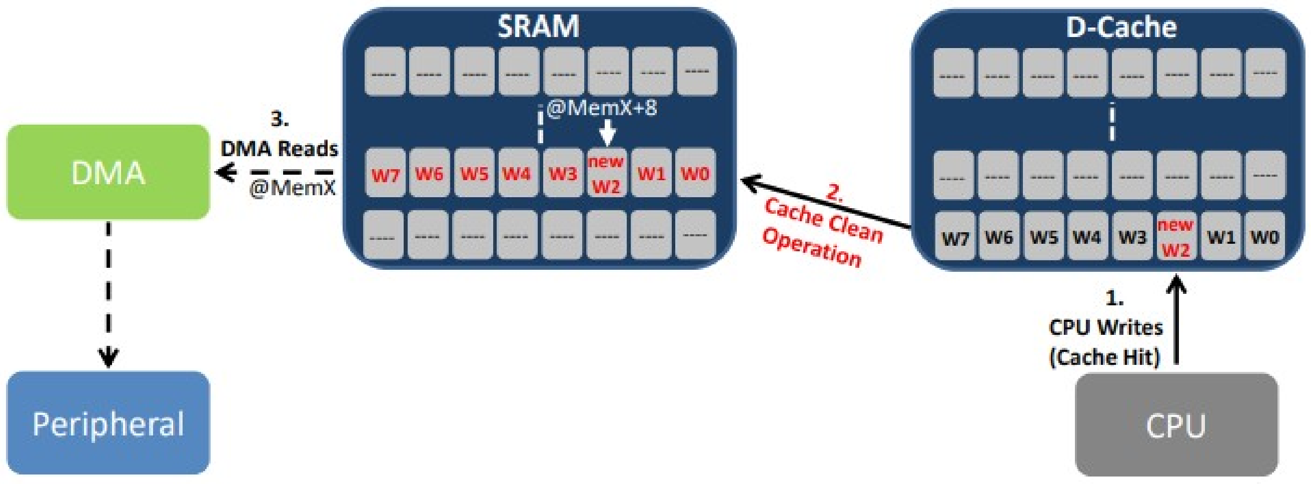
2. When the DMA reads the transmit buffer, it reads the old value present in the main memory and not the latest value updated by the CPU which is still in the cache.

1. The CPU writes data to the tx_buffer[] which will be transmitted by the DMA.

2. A cache clean operation is performed to flush the cached tx_buffer[] into the SRAM before enabling the DMA transfer.

3. The DMA reads from the SRAM will now be coherent.

Figure 4-3. Cache Clean Operation After CPU Writes to D-Cache


SRAM
@MemX+8
W7 W6 W5 W4 W3 new W2 W1 W0
D-Cache
W7 W6 W5 W4 W3 new W2 W1 W0
DMA
3. DMA Reads @MemX
2. Cache Clean Operation
1. CPU Writes (Cache Hit)
Peripheral
CPU

Code Showing A Cache Clean Operation after the CPU writes to D-Cache

```
int main (void)
{
      ......

    strcpy(tx_buffer, "DMA Transmit String");

    SCB_CleanDCache_by_Addr((uint32_t*)tx_buffer, DMA_TRANSFER_SIZE);

    xdmac_channel_enable(XDMAC, XDMA_CH_TX);
}
```

Source:
http://ww1.microchip.com/downloads/en/DeviceDoc/Managing-Cache-Coherency-on-Cortex-M7-Based-MCUs-DS90003195A.pdf