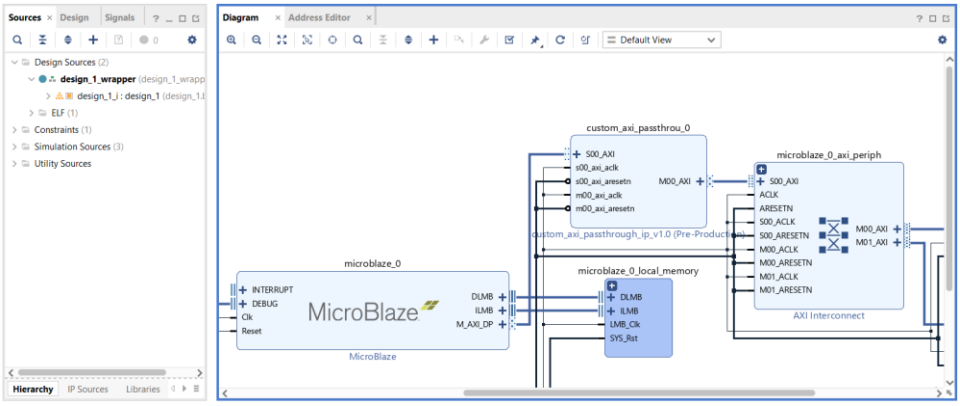
==HOW TO USE==

1. Main application circuit:



The main purpose of this component is to allow data editing while it passes the AXI bus (which could be useful in testing/debugging, for example).

The main way to use this IP is to put it in between a microprocessor (or a microcontroller) and its AXI interconnect. This way it can allow you to edit any transaction coming to (or from) any of the slaves connected to this interconnect. But theoretically you can use it (the 2-bus input version of it, this version is only setup for an MP and its interconnect) for basically any AXI-Lite master-slave connection.

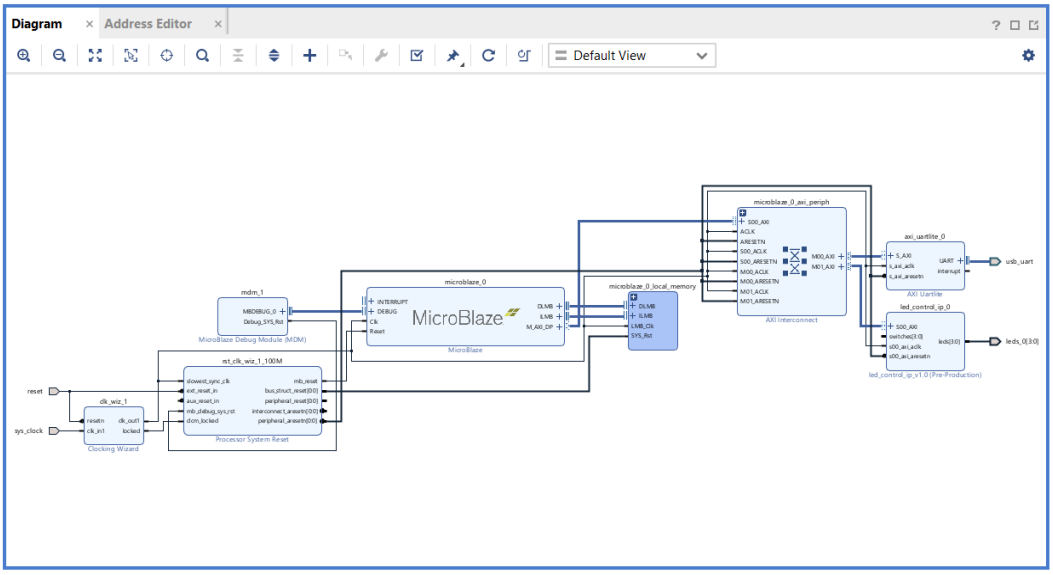
The way that it works is by replacing (or inverting) certain bits coming from (or to) a certain address. The IP has 4 main registers for each parameter set (those are separate for reading and writing transactions):

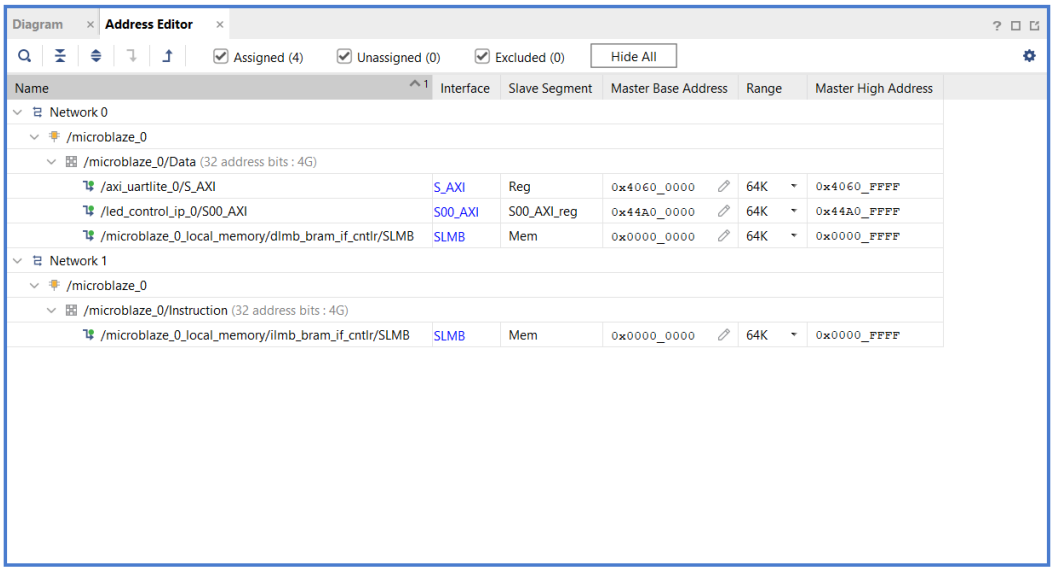
- Read or write replacement address;
- Read or write replacement value;
- Read or write replacement mask1 (allows to choose which bits should be edited or not: 0 – don't touch this bit, 1 – edit this bit);
- Read or write replacement mask2 (allows to choose if a certain bit should be replaced by the same bit of the replacement value or just inverted: 0 – replace this bit with replacement value's version of it, 1 – invert this bit. Only works if the same bit of the replacement mask1 is set to 1).

There can also be several of those parameter sets (from 1 to 16), which can be setup via the UI of the IP. (Higher counts of parameter sets should be selected with caution, as all 16 parameter sets probably won't be viable at higher clock speeds. The IP was mainly designed with a clock speed of 100MHz or below in mind).

2. Proper IP setup in the UI and address editor:

An example circuit with some AXI peripherals

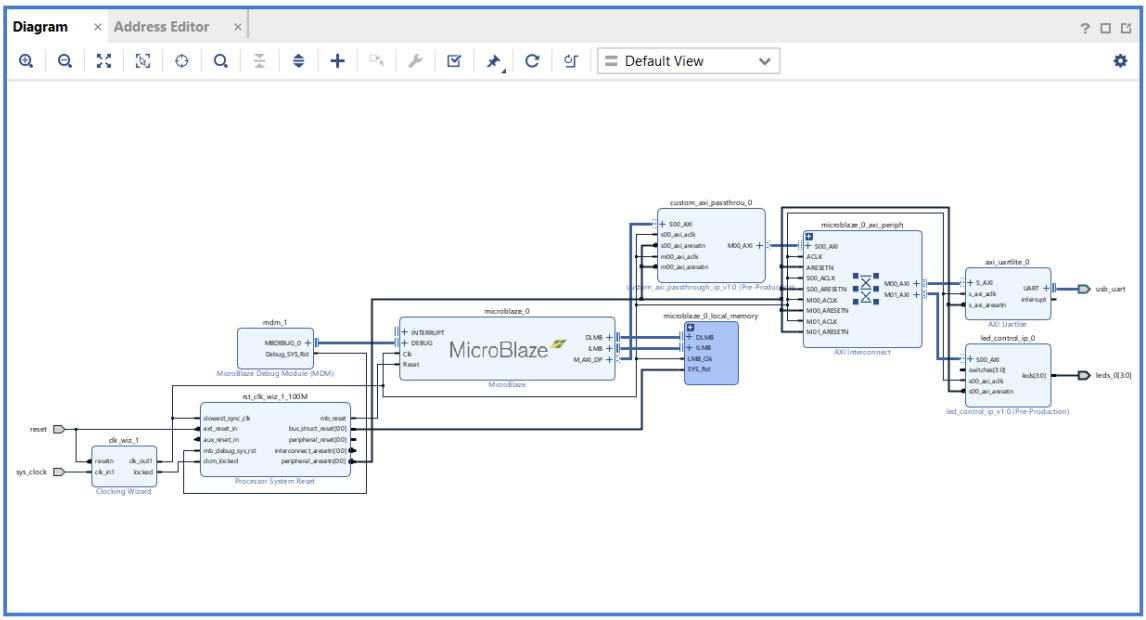


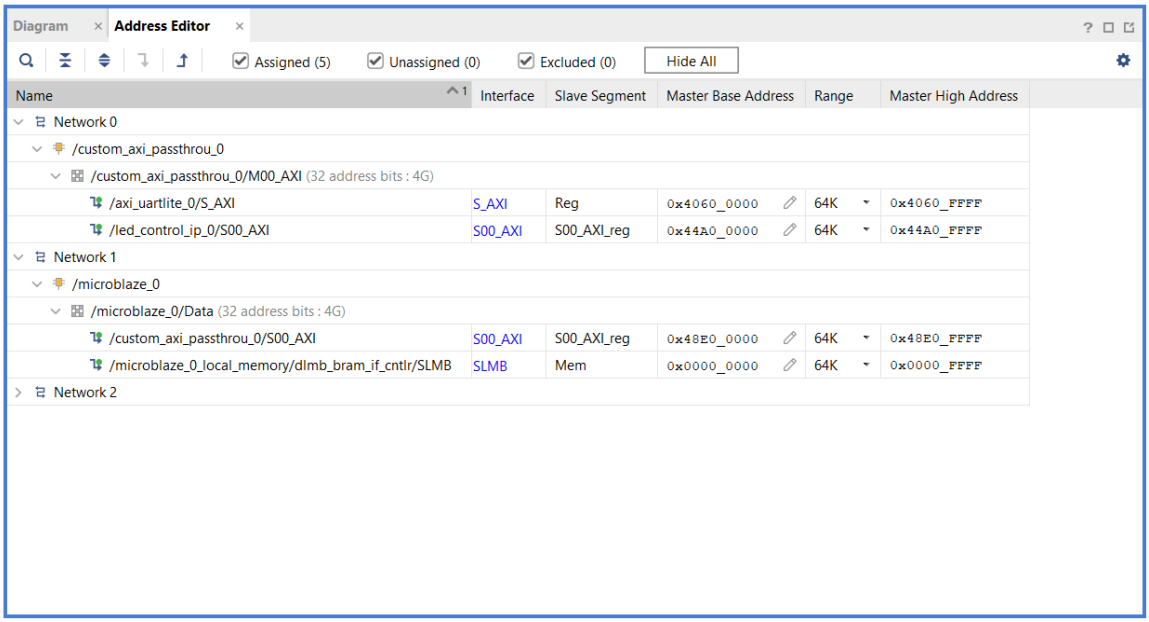


Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/microblaze_0					
/microblaze_0/Data (32 address bits : 4G)					
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/led_control_ip_0/S00_AXI	S00_AXI	S00_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network 1					
/microblaze_0					
/microblaze_0/Instruction (32 address bits : 4G)					
/microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF

Note: this is the point at which you should already (if you will eventually need to) export your XSA-file from the project (before the IP was added). This will be explained later in the tutorial.

That same circuit after proper addition of this IP



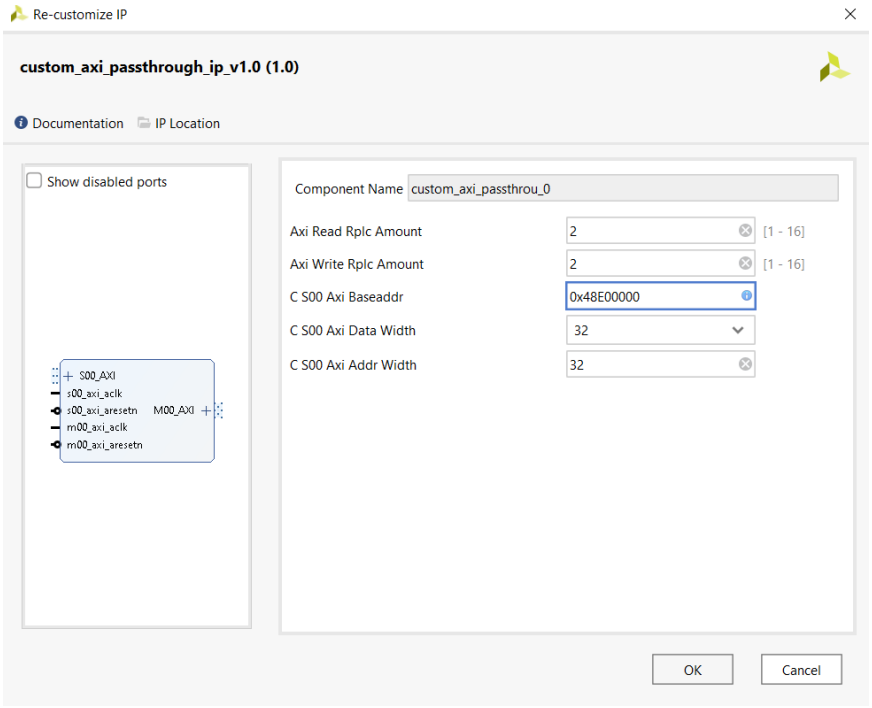


Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/custom_axi_passthrou_0					
/custom_axi_passthrou_0/M00_AXI (32 address bits : 4G)					
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/led_control_ip_0/S00_AXI	S00_AXI	S00_AXI_reg	0x44A0_0000	64K	0x44A0_FFFF
Network 1					
/microblaze_0					
/microblaze_0/Data (32 address bits : 4G)					
/custom_axi_passthrou_0/S00_AXI	S00_AXI	S00_AXI_reg	0x48E0_0000	64K	0x48E0_FFFF
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K	0x0000_FFFF
Network 2					

The addresses for the peripherals better be left the same, this can simplify the application process a bit.

The only condition for the address of the IP itself is that it (ideally) shouldn't intersect with any of the peripherals' addresses.

An example of UI setup



AXI Read&Write Rplc Amount: the amount of previously discussed (in the first section) parameter sets for either Reading or Writing transactions. It of course does affect overall performance (marginally), therefore, don't select more parameter sets than you need (especially at higher clock speeds).

If more parameter sets were selected than what's feasible at that clock speed, than some of the later parameter sets won't work (Vivado will also warn about timing issues).

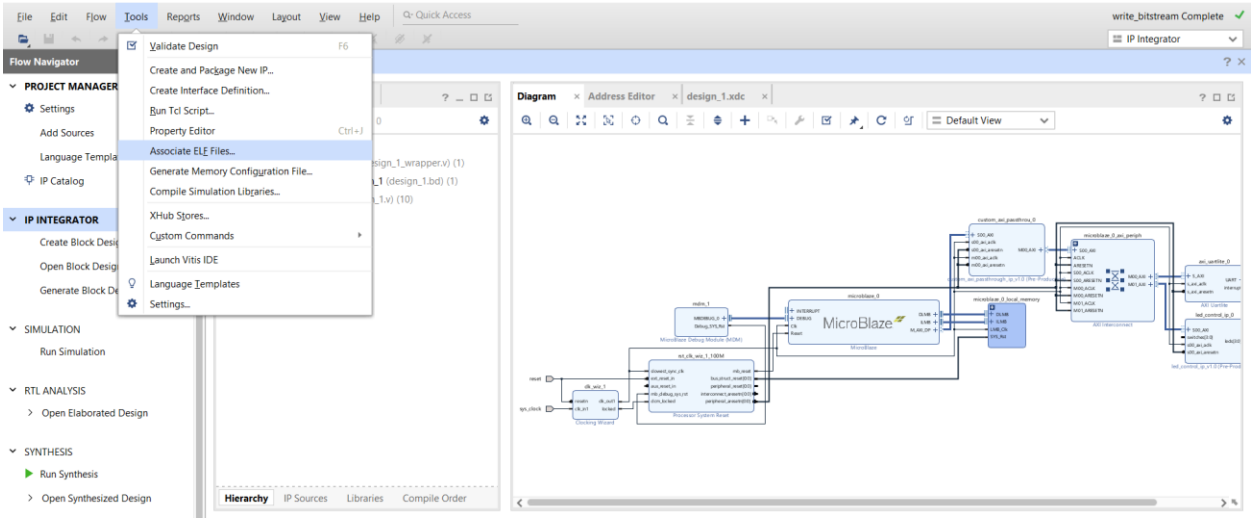
S00 AXI Baseaddr: the address which you need to write to (from a microprocessor, for example) in order to edit the parameter sets. This parameter better be set to the address of this IP (in the Address Editor). (This is also why it was mentioned earlier that the address ideally shouldn't intersect with any of the peripherals' addresses: because usually you would be writing to that address in order to program the IP.)

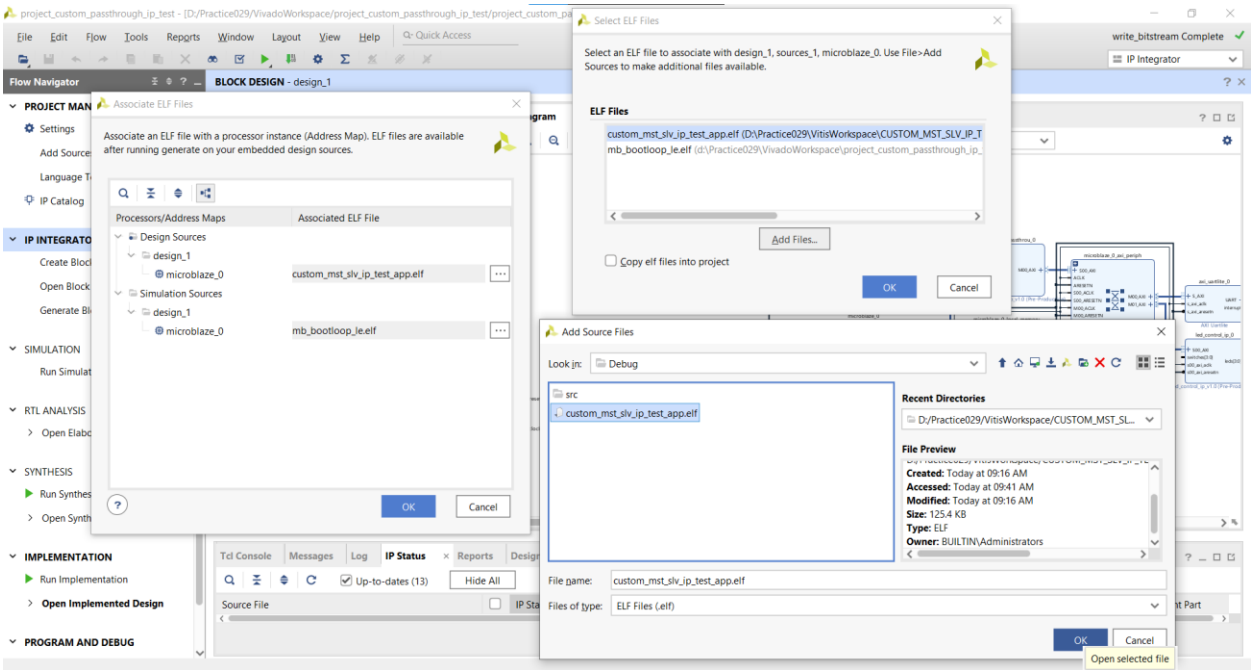
If there's a difference between the IP's address and this parameter, then you can still write to S00 AXI Baseaddr in order to edit the parameter sets. However, writing to the IP's address in this situation will not allow you to edit anything.

S00 AXI Data&Address Width: the IP currently supports 32-bit and 64-bit data width, but only 32-bit address width.

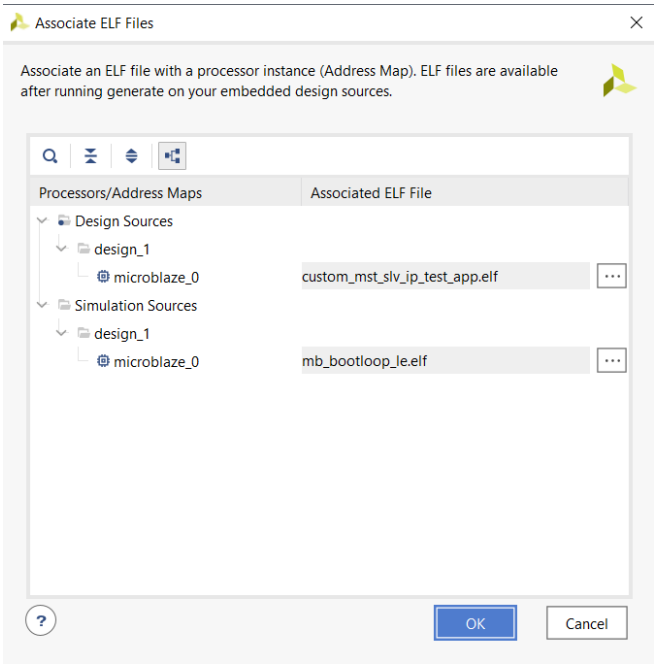
3. Working with Vitis. Programming the IP:

The main rule about working with this IP in Vitis is treating the IP as if it doesn't exist: you have to export XSA-file before the IP was added to the circuit. Otherwise, the microprocessor won't even know that there's anything connected to it (beyond this IP). You will still be able to use the generated ELF-file with your Vivado project (where the IP was already added). For that you will just need to use the «associate the ELF-files» option.





The end result should look like this

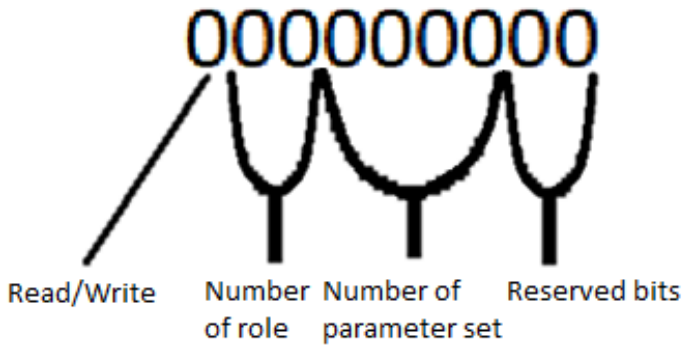


Programming procedure:

An example of programming write address, value and mask1 to the IP

```
54 int main()
55 {
56     init_platform();
57
58      uint64_t address = 0x48E00000;
59
60
61     // Setup of write address
62
63     Xil_Out32(0x48E00000 + 0b100000100, 0x44A00000);
64
65     // Setup of write value
66
67     Xil_Out16(0x48E00000 + 0b101000100, 0b100001001);
68
69     // Setup of write mask
70
71     Xil_Out32(0x48E00000 + 0b110000100, 0xFFFFFFFF);
72
```

First 9 bits of the address (except for 2 least significant ones, those are only there because they are reserved for special AXI actions, and therefore can't be edited directly) allow you to choose what specifically you want to edit with this transaction (read or write; address, value, mask1 or mask2; number of the parameter set):



Number of parameter set: 0000 – 1111 (0 - 15). You should choose in the UI the maximum amount of those sets (for either reading or writing transactions).

Number of role: 00 – address, 01 – value, 10 – mask1, 11 – mask2 (work of these registers is explained in the first paragraph).

Read/Write: 0 – for reading transactions, 1 – for writing transactions.

In the example the IP was setup to edit the data each time there is a writing transaction with the address of 0x44A00000. The 1st mask was set to 0xFFFFFFFF, meaning all bits of this data will be edited. The 2nd mask was left unedited (0x00000000), meaning none of the bits that should be edited will be inverted (instead, all the bits that should be edited will be taken from the replacement value). The replacement value was set to 0b100000000 (considering the masks, it means that each time the data needs to be edited, it will just be replaced with this value). All of those parameters were written to the 2nd parameter set.

The calculation in general is (not including checking if the address matches):

Out = (InitialData & ~Mask1) | (ReplacementValue & Mask1 & ~Mask2) | (~InitialData & Mask1 & Mask2)