Memory-Mapped I/O Demo

To create the file type the command vi task10.c

Now write this code (to write the code enter I in the keyboard)

#include <stdint.h>

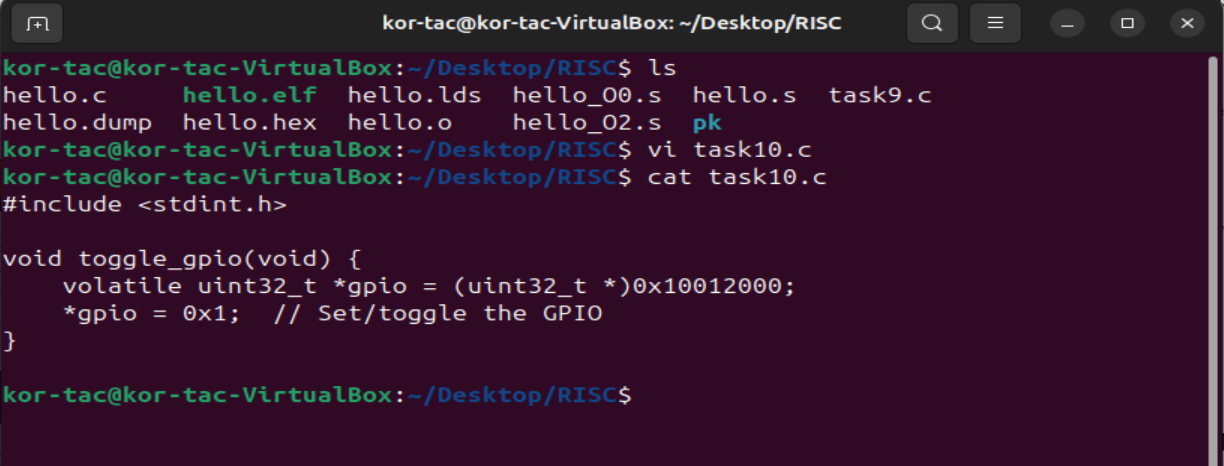
void toggle\_gpio(void) {

volatile uint32\_t \*gpio = (uint32\_t \*)0x10012000;

\*gpio = 0x1; // Set/toggle the GPIO

}

Then press esc then enter :wq tow write into the file and exit it.



**volatile**

* **Why we need volatile:**
  + Tells the compiler **"this memory location can change at any time and must not be optimized out."**
  + Without volatile, the compiler might assume that writing \*gpio = 0x1; has no side effects (especially if it doesn’t see the value being used again) and **optimize it away** during compilation.
  + But for **memory-mapped I/O**, like GPIO, **writes have side effects** — they trigger hardware actions.

🔁 So, volatile **forces** the compiler to always perform the store operation exactly as written.

**✅ Memory Alignment**

* gpio is a pointer to a uint32\_t (32-bit unsigned integer).
* **Alignment** means that the address 0x10012000 must be aligned to a 4-byte boundary (which it is).
  + That means the address must be a multiple of 4.
* **Why alignment matters:**
  + Most processors (including RISC-V) expect 32-bit data to be aligned on 4-byte boundaries.
  + **Unaligned access** may cause faults or incorrect data reads/writes.

| **Concept** | **Purpose / Explanation** |
| --- | --- |
| volatile | Prevents compiler from removing or reordering the store |
| uint32\_t | Ensures the access is 32 bits (matches hardware register) |
| Alignment | 0x10012000 is 4-byte aligned, safe for 32-bit access |