Project Based Learning:

STM32G0B1CBT6

# Project Overview:

Describe the project here. What will students do? What will students learn? How will students present their project?

# Standards:

* List learning and content standards here
* List learning and content standards here
* List learning and content standards here

# Objectives:

* List objectives here
* List objectives here
* List objectives here

# Requirements/Task(s):

Task 1 Outline the project requirements. What task(s) should the students complete?

Task 2

Task 3

# Record your notes/research here:

This is where students can record their ideas and research as they gather the information needed to complete their project.

# Outline the steps/plan for your project:

After students complete any research necessary, this is where they will create a plan for their project. Consider requiring teacher approval before they continue to the creation/implementation phase of the project.

## Teacher initials \_\_\_\_\_\_\_\_\_\_\_\_

### You are ready to create your project! Please revisit the project tasks/requirements as you work.

# Summarize what you learned:

Possible student prompts: What did you learn? What worked well? What was the most challenging aspect of this project? What will you do differently next time?

# Add the link to your project here:

Link to access project

### Required Question and Their Answer from STM32G0B1CBT6

### ✅ ****1. Core Architecture Basics****

* ARM Cortex-M family kya hota hai?
* Cortex-M0 aur Cortex-M0+ mein kya farak hai?

"Cortex-M0+ is like an upgraded Cortex-M0 — it gives better power, faster I/O, and simpler pipeline, making it more efficient for low-power embedded applications."

Cortex-M0+ is a better version of M0 — it gives more speed and uses less power.

* Ye kis ISA (Instruction Set Architecture) ko support karta hai? → (ARMv6-M, Thumb instruction set)
* Kitne-bit ka core hai? → 32-bit
* Kitne stage ka pipeline hai? → 2-stage pipeline in Cortex-M0+

### ✅ ****2. Registers and Operation****

* Kitne general-purpose registers hote hain? → 13 GPRs (R0–R12), plus SP, LR, PC
* **Program Counter (PC)**, **Stack Pointer (SP)**, **Link Register (LR)** ka role kya hota hai?
* Interrupt ke time kaunse registers automatically save hote hain?

### Summary in simple words:

* Total **8 registers** save hote hain: R0-R3, R12, LR, PC, xPSR
* Ye **hardware automatically save** karta hai interrupt entry pe
* Return hone pe ye values **automatically restore** ho jaati hain
* PSR (Program Status Register) ka kaam kya hai?

PSR ek 32-bit status register hai jo flags, interrupt info, aur CPU mode ke baare me data rakhta hai. Ye teen parts me divided hota hai – APSR, IPSR, aur EPSR. Mostly use hota hai arithmetic result check karne (like zero, negative, overflow) aur interrupt handle karne me.

PSR 3 part me divide hota hai – APSR, IPSR, EPSR. APSR me arithmetic flags hote hain, IPSR me current interrupt number hota hai, aur EPSR CPU execution state jaise Thumb mode track karta hai. Ye teenon milkar CPU ke current status ko define karte hain.

### ✅ ****3. NVIC (Nested Vectored Interrupt Controller)--Done****

* NVIC kya karta hai?
* Cortex-M0+ mein maximum kitne interrupts supported hote hain?
* Preemption and priority kaise kaam karta hai?

### ✅ ****4. Exception and Interrupts****

* ARM mein exception handling kaise hoti hai?
* HardFault, NMI, SysTick kya hote hain?
* Stack frame kaisa dikhta hai interrupt ke time?

### ✅ ****5. Memory and Bus****

* Cortex-M0+ kis memory bus ko support karta hai? → AHB-Lite
* Little endian vs Big endian?
* Harvard architecture kya hota hai?

### ✅ ****6. Power and Performance****

* Cortex-M0+ kis tarah power efficient hai?
* Low power modes (like sleep, deep sleep) available hai ya nahi?
* Clock gating kya hota hai?

### ✅ ****7. Debugging and Tools****

* SWD (Serial Wire Debug) kya hota hai? (Cortex-M0+ mein JTAG nahi hota, SWD hota hai)
* CMSIS kya hota hai?
* Linker script mein memory define kaise karte hain?

### ✅ ****8. Practical Questions (MCU-specific)****

* Tumhare project mein Cortex-M0+ ka kaunsa MCU use hua hai?
* Tumne kis IDE/toolchain ka use kiya (STM32CubeIDE, Keil, etc.)?
* Timer, ADC, UART, DMA jaise peripherals ka access kaise hota hai core ke through?
* ISR (Interrupt Service Routine) likhne ka syntax kya hai?

### 9.🧠 ****Typical Interview Questions**** (Examples):

1. Cortex-M0+ kitne-bit ka core hai? Uska instruction set kya hai?
2. Cortex-M0+ mein interrupt handling kaise hoti hai? Stack frame mein kya store hota hai?
3. NVIC ka kya kaam hota hai? Tumne uska use kaise kiya?
4. Thumb instruction set kya hota hai? Kya ye 32-bit instruction support karta hai?
5. Cortex-M3 aur M0+ mein kya farak hai?

### 🔍 Extra Tip:

Agar tum **STM32** series use kar rahe ho (jaise STM32G0B1CBT6), to uske **Reference Manual** se related questions bhi aate hain – jisme Cortex-M0+ core hote hue peripherals ka use kaise hota hai, yeh bhi included hota hai.

### 🔚 Summary – ****Interview ke liye kya yaad rakho:****

| **Topic** | **Importance** |
| --- | --- |
| Core architecture & registers | ⭐⭐⭐⭐ |
| NVIC and ISR | ⭐⭐⭐⭐ |
| Exception handling | ⭐⭐⭐ |
| Memory & Bus system | ⭐⭐⭐ |
| Debugging & SWD | ⭐⭐ |
| Power-saving features | ⭐⭐ |
| Cortex-M0+ vs others | ⭐⭐ |
| Practical coding with peripherals | ⭐⭐⭐⭐ |

**What is ARM Cortex-M family?** (Interview style in points)

1. **ARM Cortex-M family** ek group hai 32-bit microcontroller cores ka, jo **ARM company** ne banaya hai.
2. Ye cores **RISC (Reduced Instruction Set Computer)** architecture par based hote hain – simple aur fast instruction set ke liye.
3. Inka use **embedded systems**, **IoT devices**, **consumer products**, aur **industrial control** me hota hai.
4. Cortex-M family me alag-alag cores hote hain:
   * **Cortex-M0 / M0+** → low power & simple applications
   * **Cortex-M3 / M4** → better performance, M4 me DSP features bhi hote hain
   * **Cortex-M7** → high performance applications ke liye
5. Ye cores **low power**, **low cost**, aur **fast interrupt response** ke liye design kiye gaye hote hain.
6. Programming inhe **CMSIS library** ke through easily kiya ja sakta hai (specially STM32 MCUs me).

**What is ARM Cortex-M family?**

1. ARM Cortex-M is a group of processors made by a company called ARM.
2. These are 32-bit processors used in small electronic devices.
3. They are made for **simple**, **low-power**, and **low-cost** systems.
4. Cortex-M is used in things like:
   * Home electronics
   * Industrial machines
   * IoT devices
   * Smart sensors
5. Different types of Cortex-M:
   * **Cortex-M0 / M0+** – very small and low power
   * **Cortex-M3 / M4** – faster and better, M4 has extra features
   * **Cortex-M7** – very fast, used in complex systems
6. These are easy to use and fast with interrupts.
7. Many companies like **STMicroelectronics (STM32)** use Cortex-M cores in their microcontrollers.

**Q.** Ye kis ISA (Instruction Set Architecture) ko support karta hai? → (ARMv6-M, Thumb instruction set)

**Cortex-M0 / M0+ ISA Summary**

* Cortex-M0 aur M0+ **ARMv6-M** instruction set architecture (ISA) ko support karte hain.
* Ye cores sirf **Thumb instructions** ko support karte hain — matlab saare instructions **16-bit** ke hote hain.
* 16-bit instructions hone se code size chhota hota hai, aur memory bhi kam use hoti hai.
* Inme **basic aur limited instructions** hote hain:
  + No divide instruction
  + No floating point support
  + No 32-bit ARM instructions (only 16-bit Thumb)
* Ye cores **low power** aur **low cost** applications ke liye perfect hote hain, jaise:
  + Sensor devices
  + Battery-powered products
  + Simple embedded controllers
* Inka design simple hota hai, isliye yeh beginners aur small embedded projects ke liye best hote hain.

### ****1. ARMv6-M kya hai?****

* **ARMv6-M** ek **Instruction Set Architecture (ISA)** hai.
* Ye ek **set of rules & instructions** ka group hai, jo batata hai ki **Cortex-M0** aur **M0+** type ke cores ka processor **kaise kaam karega**.
* Is ISA ko **ARM company** ne design kiya hai specially **low-power embedded systems** ke liye.

### ⚙️ ****2. ISA ka matlab kya hota hai?****

* **Instruction Set Architecture** ek **software-hardware bridge** hota hai.
* Ye define karta hai:
  + Kaunse instructions CPU samajh sakta hai
  + Register ka size kya hai (yaha 32-bit)
  + Memory kaise access hoti hai
  + Interrupts kaise handle hote hain

### 🧠 ****3. ARMv6-M me kya-kya hota hai?****

* **Thumb-1 Instruction Set**:
  + 16-bit short instructions — fast & compact
* **Basic Arithmetic/Logic Instructions** (add, sub, and, or, etc.)
* **Load/Store Instructions** (memory se data lena ya bhejna)
* **Branch/Jump Instructions** (program flow control)
* **Interrupt Handling** support (NVIC ke through)

### 🎯 ****4. ARMv6-M kyu banaya gaya?****

* **Simple aur low-cost MCU** ke liye.
* Jaise: sensors, small controllers, wearables, etc.
* Jisme power aur performance ka balance chahiye.

### 🧩 ****5. Kaisa dikhta hai (abstractly)?****

ARMv6-M ka koi physical shape nahi hota — ye ek **logical blueprint** hota hai jo batata hai core ko **kaise kaam karna hai**. Processor companies (jaise STM, NXP, etc.) is ISA ko use karke apna **Cortex-M0/M0+ based MCU** banate hain.

### 🔚 Summary:

ARMv6-M ek simple instruction set hai jo Cortex-M0/M0+ me use hota hai. Iska purpose hai low-power, low-cost devices me efficient aur fast processing dena. Ye Thumb instructions ke through kam memory use karta hai aur embedded world ke liye perfect hai.

**Q. Kitne general-purpose registers hote hain? → 13 GPRs (R0–R12), plus SP, LR, PC**

### 🔹 Cortex-M ke General Purpose Registers:

* Total **13 General Purpose Registers (GPRs)** hote hain.
* Ye registers ka naam hota hai **R0 se R12** tak.
* Har register 32-bit hota hai.
* Inke alawa kuch **special registers** bhi hote hain:

### 🔸 Special Registers:

* **R13 → SP (Stack Pointer)**  
  Function call ya interrupt ke time pe data store karta hai (stack ke liye).
* **R14 → LR (Link Register)**  
  Function call ke baad wapas aane ke address ko store karta hai.
* **R15 → PC (Program Counter)**  
  Abhi kaunsa instruction execute ho raha hai, uska address batata hai.

### ✅ Short Summary:

| **Register** | **Naam** | **Kaam (Purpose)** |
| --- | --- | --- |
| R0–R12 | GPRs | Data processing, temporary values |
| R13 | SP (Stack Ptr) | Stack manage karta hai |
| R14 | LR (Link Reg) | Return address rakhta hai |
| R15 | PC (Prog Counter) | Next instruction ka address rakhta hai |
| 🔸 Extra Features (Family-wise):  * **Cortex-M0 / M0+**: Basic set of registers + simple interrupt system ❌ No FPU, no advanced debug * **Cortex-M3 / M4 / M7**: Same GPRs + ✅ **Advanced interrupt system**, ✅ **FPU** (M4/M7), ✅ **Debug support** * **Cortex-M33 / M55**: Same GPRs + ✅ Security (TrustZone), ✅ FPU, ✅ DSP instructions  ✅ Summary: ✅ **R0–R12 + SP + LR + PC** → Har Cortex-M family core me **same** hote hain. 🔁 Difference aata hai extra features (like FPU, debug, DSP, security) me.  ***Q.PSR (Program Status Register) ka kaam kya hai?*** ✅ 1. APSR (Application Program Status Register) **Use:**   * Arithmetic operation ke **flags** store karta hai:   + N → Negative flag   + Z → Zero flag   + C → Carry flag   + V → Overflow flag * Ye flags conditional branching (like if-else) ya math result check karne ke kaam aate hain.   📌 **Example:** Addition ke baad agar result zero ho, toh Z=1 ho jata hai. ✅ 2. IPSR (Interrupt Program Status Register) **Use:**   * Batata hai **kaunsa interrupt service routine (ISR)** abhi chal raha hai. * Agar no interrupt → value 0 * Agar interrupt active → us interrupt ka number store hota hai.   📌 **Example:** Agar EXTI0 interrupt chal raha hai → IPSR = interrupt number of EXTI0 ✅ 3. EPSR (Execution Program Status Register) **Use:**   * CPU ki **execution state** dikhata hai. * Mostly **Thumb state** ke control ke liye use hota hai. * Also holds some **IT (If-Then)** instruction state bits.   📌 **Note:** ARM Cortex-M hamesha **Thumb instruction set** use karta hai → EPSR ensure karta hai ki CPU Thumb mode me hi rahe. 🔸 Interview Summary: **PSR 3 part me divide hota hai – APSR, IPSR, EPSR. APSR me arithmetic flags hote hain, IPSR me current interrupt number hota hai, aur EPSR CPU execution state jaise Thumb mode track karta hai. Ye teenon milkar CPU ke current status ko define karte hain.** ✅ ****3. NVIC (Nested Vectored Interrupt Controller)--Done****  * NVIC kya karta hai? * Cortex-M0+ mein maximum kitne interrupts supported hote hain? * Preemption and priority kaise kaam karta hai?  ✅ 1. ****NVIC kya karta hai?****  * NVIC is the **interrupt controller** used in Cortex-M processors. * Iska kaam hai:   + Interrupt ko detect karna.   + Har interrupt ko ek **priority** dena.   + Fast response ke liye **vector table** se direct jump karwana. * Ye **nested interrupts** ko support karta hai (matlab high-priority interrupt low-priority ko interrupt kar sakta hai).   📌 **Simple Example:** Agar Timer aur UART dono interrupt de rahe ho, NVIC dekh ke decide karega kaun pehle chalega (priority ke base par). ✅ 2. ****Cortex-M0+ mein maximum kitne interrupts supported hote hain?****  * Cortex-M0+ supports:   + **Up to 32 external interrupts** (IRQ0 to IRQ31)   + Plus **16 system exceptions** (like Reset, NMI, HardFault, etc.)   📌 **Note:** Ye **implementation-specific** hota hai, toh exact number chip par depend karta hai (like STM32G0 me alag ho sakta hai). ✅ 3. ****Preemption and Priority kaise kaam karta hai?****  * NVIC assigns **priority** to each interrupt. * **Lower value = higher priority** (priority 0 is highest). * **Preemption:**   + Agar ek high-priority interrupt aata hai jab low-priority interrupt chal raha ho, toh high waala usko **interrupt kar sakta hai** (ye nested interrupt hai). * Priority bits limited hote hain:   + Cortex-M0+ usually supports **up to 4 levels** (2 bits).   📌 **Example:** UART interrupt priority 1, Timer priority 3 → UART timer ko preempt kar sakta hai. 🔸 Interview Summary Statement: **NVIC Cortex-M processors me interrupt manage karta hai. Ye har interrupt ko priority deta hai aur nested interrupts ko support karta hai. Cortex-M0+ me up to 32 external interrupts hote hain, aur priority-based preemption system se fast aur efficient response possible hota hai.** ✅ ****5. Memory and Bus****  * Cortex-M0+ kis memory bus ko support karta hai? → AHB-Lite * Little endian vs Big endian? * Harvard architecture kya hota hai?  ✅ ****1. Cortex-M0+ kis memory bus ko support karta hai?**** **Answer:**   * **Cortex-M0+** core **AHB-Lite (Advanced High-performance Bus-Lite)** interface ko support karta hai. * Ye bus **simplified version hai AHB ka**, specially low-power aur low-cost MCUs ke liye design kiya gaya hai. * Iska use processor aur memory/peripherals ke beech **data transfer** ke liye hota hai.   **Key points (interview style):**   * AHB-Lite = Single master bus (MCU core hi master hota hai) * Fast and simple communication between CPU and memory * Support karta hai 32-bit data width  ✅ ****2. Little Endian vs Big Endian**** **Answer:**   * Ye **data store karne ka tarika** hota hai memory me (multi-byte data like int, float).  🔹 Little Endian:  * **Least significant byte (LSB)** pehle store hota hai (lower address pe). * Example: 0x12345678 will be stored as → 78 56 34 12  🔹 Big Endian:  * **Most significant byte (MSB)** pehle store hota hai (lower address pe). * Example: 0x12345678 will be stored as → 12 34 56 78   **Cortex-M cores (like M0+) → Little Endian by default** ✅ ****3. Harvard Architecture kya hota hai?**** **Answer:**   * Harvard architecture me **code aur data ke liye alag-alag memory buses** hote hain. * Iska matlab: **CPU ek hi time me instruction fetch aur data access dono kar sakta hai**.   **Key points:**   * Separate buses = faster execution * Common in microcontrollers like STM32, Cortex-M * Opposite of **Von Neumann architecture** (jisme data/code same bus use karte hain)  ✅ ****Von Neumann Architecture kya hota hai?**** **Definition:**   * Ye ek computer architecture hai **jisme program instructions aur data dono ek hi memory aur bus system me store hote hain.**  🔹 ****Key Points:****  * **Single memory** for both **code (instructions)** and **data** * **Single bus** used for both **data fetch** and **instruction fetch** * **Slower than Harvard architecture**, because code aur data ek time pe access nahi kar sakte (bus conflict hota hai) * Simple and cost-effective for basic systems * Used in **general-purpose computers** (PCs, laptops, etc.)  🔹 ****Example Use:****  * Agar ek instruction fetch kar raha hai, to same time pe data memory access nahi kar sakta. * Isliye kuch delay hota hai — **called von Neumann bottleneck**.  🔹 ****Compare with Harvard Architecture:****  | **Feature** | **Von Neumann** | **Harvard** | | --- | --- | --- | | Memory for code/data | Shared (same) | Separate | | Bus for code/data | Same | Separate | | Speed | Slower (bottleneck) | Faster | | Cost/Complexity | Simpler, cheaper | Complex but faster |  ✅ ****8. Practical Questions (MCU-specific)****  * Tumhare project mein Cortex-M0+ ka kaunsa MCU use hua hai? * Tumne kis IDE/toolchain ka use kiya (STM32CubeIDE, Keil, etc.)? * Timer, ADC, UART, DMA jaise peripherals ka access kaise hota hai core ke through? * ISR (Interrupt Service Routine) likhne ka syntax kya hai?   ✅ **8. Practical Questions (MCU-specific)** – Interview style in **simple Hinglish + points** 🔹 ****1. Tumhare project mein Cortex-M0+ ka kaunsa MCU use hua hai?****  * **Example answer**: Mere project mein maine STM32G0B1CBT6 MCU use kiya hai.   + Ye Cortex-M0+ core based MCU hai.   + Iska use maine ek smart battery charger ke firmware development me kiya.  🔹 ****2. Tumne kis IDE/toolchain ka use kiya?****  * **Example answer**: Maine mostly ***STM32CubeIDE*** ka use kiya hai.   + Ye Eclipse-based IDE hai jo code likhne, compile karne, aur debug karne ke liye use hoti hai.   + STM32CubeMX bhi use kiya for peripheral configuration and code generation.   + Kabhi-kabhi **Keil uVision** bhi use kiya chhote test projects ke liye.  🔹 ****3. Timer, ADC, UART, DMA jaise peripherals ka access kaise hota hai core ke through?****  * Cortex-M0+ core ke saath **peripherals** ko access kiya jata hai through **memory-mapped registers**. * Har peripheral ka **register address fixed hota hai**, jo **AHB-Lite** or **APB bus** ke through connected hota hai. * Hum C code me HAL\_ ya LL\_ libraries use karke peripheral control karte hain.   + Example: HAL\_TIM\_Start(), HAL\_ADC\_Start\_DMA(), HAL\_UART\_Transmit()  🔹 ****4. ISR (Interrupt Service Routine) likhne ka syntax kya hai?****  * ISR likhne ke liye STM32 me usually aise likhte hain:   c  CopyEdit  void TIM6\_DAC\_IRQHandler(void)  {  HAL\_TIM\_IRQHandler(&htim6); // Timer interrupt handle  }  Ya agar HAL\_TIM\_PeriodElapsedCallback() use kar rahe ho, to:  c  CopyEdit  void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)  {  if(htim->Instance == TIM6)  {  // Your timer interrupt code  }  }  ✅ **8. Practical Questions (MCU-specific)** – Answer in **basic English + points** 🔹 1. Which MCU did you use with Cortex-M0+ in your project?  * I used **STM32G0B1CBT6** MCU. * It has **Cortex-M0+ core** inside. * I used it for making a **battery charger firmware**.  🔹 2. Which IDE or toolchain did you use?  * I used **STM32CubeIDE**.   + It is used for writing code, compiling, and debugging.   + It also has **STM32CubeMX** for peripheral setup. * Sometimes I also used **Keil uVision** for testing small programs.  🔹 3. How do you access peripherals like Timer, ADC, UART, DMA from the core?  * Peripherals are connected to the core using **memory-mapped registers**. * Each peripheral has its own register address. * I used **HAL drivers** to control them.   + Example: HAL\_TIM\_Start() for timer, HAL\_ADC\_Start\_DMA() for ADC, HAL\_UART\_Transmit() for UART.  🔹 4. What is the syntax of writing ISR (Interrupt Service Routine)?  * Example for timer interrupt:   c  CopyEdit  void TIM6\_DAC\_IRQHandler(void)  {  HAL\_TIM\_IRQHandler(&htim6); // handle timer interrupt  }   * Or use this function for logic:   c  CopyEdit  void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)  {  if(htim->Instance == TIM6)  {  // Your interrupt code  }  } ✅ ****Startup File in Embedded Systems (Simple Explanation)**** Startup file MCU me ek **important file** hoti hai jo system ke **boot hone** par sabse pehle kaam karti hai. 🔹 ****1. Isme kya hota hai?**** Startup file usually .s (assembly) ya .c file hoti hai, aur ye cheezein contain karti hai:   * **Vector Table** → Sab interrupts ke address list (like Reset, NMI, HardFault, etc.) * **Reset Handler** → Ye function sabse pehle run hota hai power-on ya reset ke baad. * **Default ISR Functions** → Agar koi interrupt define nahi kiya, to default function call hota hai. * **Memory Initialization** → .data section RAM me copy hoti hai (from flash) → .bss section (zero-initialized data) ko clear karta hai * **Main() Function Call** → Sab kuch set karne ke baad, main() ko call karta hai.  🔹 ****2. Startup File ka kya kaam hai?****  * MCU ko **boot process** ke liye tayar karta hai. * Interrupts ka **address table** banata hai. * **Main application** tak pahunchata hai (via main()). * RAM ko sahi tareeke se initialize karta hai.  🔹 ****3. Example Startup Flow****  1. MCU power ON 2. **Reset Handler** run hota hai 3. RAM initialize hoti hai (data, bss) 4. SystemInit() call hota hai 5. **main() function** start hota hai  ✅ STM32 Startup File (usually: startup\_stm32xxxx.s) Startup file mostly **Assembly language** me hoti hai (.s extension), lekin iska structure samajhna easy hai. 🔹 1. ****Vector Table Declaration**** assembly  CopyEdit  .section .isr\_vector,"a",%progbits  .type g\_pfnVectors, %object  .size g\_pfnVectors, .-g\_pfnVectors  g\_pfnVectors:  .word \_estack ; Initial Stack Pointer  .word Reset\_Handler ; Reset Handler  .word NMI\_Handler ; NMI Handler  .word HardFault\_Handler ; Hard Fault Handler  ...  🧠 **Meaning**:   * Ye **Interrupt Vector Table** hai. * Sabse pehle \_estack: stack pointer ki starting location. * Fir sab interrupts ke function address diye jaate hain. * Example: Reset\_Handler, NMI\_Handler, HardFault\_Handler, etc.  🔹 2. ****Default Handlers**** assembly  CopyEdit  .weak NMI\_Handler  .thumb\_set NMI\_Handler, Default\_Handler  🧠 **Meaning**:   * Agar NMI\_Handler define nahi hua user code me, to Default\_Handler use hoga. * Default\_Handler ek infinite loop hota hai — MCU yahi pe rukta hai.   assembly  CopyEdit  Default\_Handler:  b . 🔹 3. ****Reset\_Handler Function**** assembly  CopyEdit  Reset\_Handler:  ldr r0, =\_sdata  ldr r1, =\_edata  ldr r2, =\_sidata  loop\_copy\_data:  cmp r0, r1  ittt lt  ldrlt r3, [r2], #4  strlt r3, [r0], #4  blt loop\_copy\_data  🧠 **Meaning**:   * Flash se **.data** section ko RAM me copy karta hai.   assembly  CopyEdit  ldr r0, =\_sbss  ldr r1, =\_ebss  movs r2, #0  🧠 **Meaning**:   * RAM me **.bss** section ko zero se fill karta hai (zero-initialized vars).   assembly  CopyEdit  bl SystemInit  bl main  🧠 **Meaning**:   * SystemInit() → Clock config, low-level settings * main() → Tumhara actual application start hota hai  🔚 Summary (Simple Flow)  1. Vector Table define hota hai 2. Default interrupt handlers set hote hain 3. Reset\_Handler start hota hai 4. .data RAM me copy hoti hai 5. .bss zero hoti hai 6. SystemInit() run hota hai 7. main() function start hota hai  ✅ ****Vector Table kya hota hai?**** (Basic English + Interview style)🔹 ****1. Definition – What is Vector Table?****  * **Vector table** ek **fixed memory location** par hoti hai jahan **interrupts aur exceptions** ke addresses (function pointers) stored hote hain. * MCU jab **reset** hoti hai ya koi **interrupt** aata hai, to processor vector table me se **function ka address read karta hai** aur wahan jump karta hai.  🔹 ****2. Vector Table me kya kya hota hai?****  * Ye ek **array of pointers** hoti hai:   + Sabse pehle: **Initial Stack Pointer (SP)**   + Fir: **Reset Handler**, **NMI**, **HardFault**, etc.   + Uske baad: **external interrupt handlers** (like Timer, UART, GPIO, etc.)   c  CopyEdit  Example:  0x0000\_0000 → Initial SP  0x0000\_0004 → Reset\_Handler  0x0000\_0008 → NMI\_Handler  0x0000\_000C → HardFault\_Handler  ... 🔹 ****3. Vector Table ki location kaha hoti hai?****  * Usually, **start of flash memory (0x00000000)** par hoti hai. * ARM Cortex MCUs me SCB->VTOR register se vector table ki location set/change ki ja sakti hai (except M0/M0+ jahan fixed hoti hai).  🔹 ****4. Vector Table ka kaam?****  * **Reset ke time**: MCU sabse pehle vector table ka **initial SP** aur **reset handler** read karta hai. * **Interrupt ke time**: MCU vector table se **interrupt service routine (ISR)** ka address fetch karta hai.  🔹 ****5. Interview Summary Answer:**** **"Vector table ek fixed memory table hoti hai jisme interrupt aur exception handlers ke address store hote hain. Jab MCU reset hoti hai ya koi interrupt aata hai, processor vector table se correct function ka address leta hai aur wahan jump karta hai. Ye table MCU ke flash memory ke starting me hoti hai."** |  |  |

### ✅ ****7. Debugging and Tools****

* SWD (Serial Wire Debug) kya hota hai? (Cortex-M0+ mein JTAG nahi hota, SWD hota hai)
* CMSIS kya hota hai?
* Linker script mein memory define kaise karte hain?

### 🔹 ****1. SWD (Serial Wire Debug) kya hota hai?****

* **SWD** means **Serial Wire Debug**.
* Ye **2-pin** debugging interface hai (SWDIO & SWCLK).
* **JTAG** ke jagah use hota hai in Cortex-M0+ (kyunki M0+ me JTAG nahi hota).
* Isse hum:
  + MCU me code flash kar sakte hain.
  + Code ko real-time me debug kar sakte hain (breakpoints, step-by-step run).
* Ye **faster aur lightweight** hota hai compared to JTAG.
* Use karte hain ST-Link, J-Link, ya CMSIS-DAP jaise programmers/debuggers ke sath.

### 🔹 ****2. CMSIS kya hota hai?****

* **CMSIS** = **Cortex Microcontroller Software Interface Standard**
* ARM ne banaya hai to make development easy for Cortex-M cores.
* Ye ek **library + header files** ka set hota hai.
* CMSIS provide karta hai:
  + Core registers ke access ke liye macros/functions.
  + NVIC, SysTick, SCB jaise peripherals ke access ke liye predefined names.
  + Startup file aur interrupt handlers ke standard naming.
* CMSIS ke parts:
  + **CMSIS-Core** (core access)
  + **CMSIS-DSP** (digital signal processing)
  + **CMSIS-RTOS** (real-time OS interface)
  + **CMSIS-Driver** (peripheral drivers standard)

### 🔹 ****3. Linker Script me Memory kaise define karte hain?****

* Linker script batata hai:
  + Code, variables, stack, etc. **MCU ke memory** me **kaha rakha jaye**.
* Ye file usually .ld extension me hoti hai (STM32CubeIDE me auto-generated hoti hai).
* Example block in .ld file:

ld

CopyEdit

MEMORY

{

FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 64K

RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 20K

}

🧠 **Explanation**:

* FLASH: Program (code) yahan rakha jata hai.
* RAM: Variables, stack, heap, etc. yahan rakhe jate hain.
* ORIGIN = Start address.
* LENGTH = Size of memory.

### 🔹 ****Linker Script kya hota hai?**** (Simple English)

**Linker Script** ek text file hoti hai jo batati hai:

❝ MCU ke memory (FLASH, RAM) me humara code, data, variables kaha store kiya jaye ❞

### 🔸 Use of Linker Script:

1. MCU ke **memory layout** ko describe karta hai.
2. Code (.text), variables (.data, .bss), stack, heap – sabke **locations** set karta hai.
3. Compiler aur linker ko help karta hai final binary banane me.

### 🔸 Common file name:

* .ld (Linker Description)
* Example: STM32G0B1CBTx\_FLASH.ld

### 🔸 Example:

ld

CopyEdit

MEMORY

{

FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 64K

RAM (xrw): ORIGIN = 0x20000000, LENGTH = 20K

}

🧠 **Meaning**:

* **FLASH**: Code store hoga 0x08000000 se start hoke 64KB tak.
* **RAM**: Variables, stack, etc. start hoga 0x20000000 se aur 20KB long hoga.

### 🔸 Without Linker Script?

* Program galat jagah jaa sakta hai.
* MCU crash ya unpredictable behavior dikhayega.

### 9.🧠 ****Typical Interview Questions**** (Examples):

1. Cortex-M0+ kitne-bit ka core hai? Uska instruction set kya hai?
2. Cortex-M0+ mein interrupt handling kaise hoti hai? Stack frame mein kya store hota hai?
3. NVIC ka kya kaam hota hai? Tumne uska use kaise kiya?
4. Thumb instruction set kya hota hai? Kya ye 32-bit instruction support karta hai?
5. Cortex-M3 aur M0+ mein kya farak hai?

### ✅ 1. ****Cortex-M0+ kitne-bit ka core hai? Uska instruction set kya hai?****

* Cortex-M0+ is a **32-bit core**.
* It uses **ARMv6-M architecture**.
* It supports **Thumb (16-bit)** instruction set.
* Some instructions are **32-bit Thumb-2**, but M0+ supports only a **subset** (not full Thumb-2).

### ✅ 2. ****Cortex-M0+ mein interrupt handling kaise hoti hai? Stack frame mein kya store hota hai?****

* Jab interrupt aata hai, MCU **automatically push karta hai** kuch registers stack me:
  + R0, R1, R2, R3
  + R12
  + LR (Link Register)
  + PC (Program Counter)
  + xPSR (Status Register)
* Fir ISR (Interrupt Service Routine) run hota hai.
* Jab ISR khatam hota hai, ye sab wapas **pop** ho jata hai.

### ✅ 3. ****NVIC ka kya kaam hota hai? Tumne uska use kaise kiya?****

* **NVIC** means Nested Vectored Interrupt Controller.
* Ye interrupts ko **enable, disable**, aur **priority** assign karta hai.
* M0+ me maximum **32 external interrupts** support hote hain.
* Main ne NVIC ka use kiya to:
  + Timer, UART, ADC interrupts **enable** kiye.
  + HAL\_NVIC\_EnableIRQ() jaise function use kiye.

### ✅ 4. ****Thumb instruction set kya hota hai? Kya ye 32-bit instruction support karta hai?****

* Thumb ek **compressed instruction set** hai, jisme mostly **16-bit instructions** hote hain.
* Cortex-M0+ only **Thumb (16-bit)** instructions ko support karta hai.
* **32-bit Thumb-2** instructions sirf higher cores jaise Cortex-M3/M4 use karte hain.

### ✅ 5. ****Cortex-M3 aur M0+ mein kya farak hai?****

| **Feature** | **Cortex-M0+** | **Cortex-M3** |
| --- | --- | --- |
| Architecture | ARMv6-M | ARMv7-M |
| Performance | Low power, simple | Higher performance |
| Instruction Set | Thumb (16-bit only) | Thumb + Thumb-2 |
| FPU Support | ❌ (No) | ✅ (Optional) |
| Interrupts | Up to 32 | Up to 240 |
| NVIC Priority Bits | 2 or 3 bits | Up to 8 bits |