Embedded Systems Interview Questions

GOING Basic

Part 1: C & Data Structures

1. What are the key differences between C and C++?
2. Explain the significance of volatile and const keywords.
3. How does #include work in C?
4. What is the difference between malloc(), calloc(), and realloc()?
5. Explain pointer arithmetic with an example.
6. What is a dangling pointer? How can it be avoided?
7. How does recursion work in C? What are its limitations?
8. What is the difference between struct and union?
9. Explain memory alignment and padding in structures.
10. What are function pointers? Give a practical use case.
11. How does typedef differ from #define?
12. Explain the difference between static and dynamic memory allocation.
13. What is a memory leak? How can it be detected?
14. Explain the difference between strcpy() and memcpy().
15. What is a segmentation fault? Common causes?
16. How does qsort() work in C?
17. Explain the difference between ++i and i++.
18. What is the purpose of restrict keyword in C?
19. How does va_arg work for variable arguments?
20. What is the difference between stack and heap memory?
21. Explain how a linked list differs from an array.
22. What are the advantages of a doubly linked list over a singly linked list?
23. How does a circular linked list work?
24. Explain the time complexity of insertion/deletion in different data structures.
25. What is a hash table? How is collision handled?
26. Explain the working of Bubble Sort vs Quick Sort.
27. What is the worst-case time complexity of Merge Sort?
28. How does Binary Search work?
29. What is a self-balancing BST?
30. Explain Dijkstra's Algorithm for shortest path.
31. What is a trie data structure?
32. How does dynamic programming optimize recursive problems?
33. What is the difference between BFS and DFS?
34. Explain LRU Cache implementation.
35. What is a bitmask? How is it useful?
36. Explain endianness and its impact on data storage.
37. What is a memory-mapped file?
38. How does fseek() work in file handling?
39. What is the difference between text and binary file modes?
40. How are command-line arguments parsed in C?
41. Explain the GCC compilation process (Preprocessing → Compilation → Assembly → Linking).

42. What is the role of a Makefile?
43. How does GDB help in debugging?
44. What are core dumps? How to analyze them?
45. Explain inline functions vs macros.
46. What is undefined behavior in C?
47. How does setjmp() and longjmp() work?
48. What is reentrancy in functions?
49. Explain memory corruption scenarios.
50. What are compiler intrinsics?

EXTRA:

1. What is the purpose of the volatile keyword in C?
2. How does recursion work in C? Provide an example.
3. What are the differences between structures and unions?
4. Explain bitwise operators with examples (&, |, ^, <<, >>).
5. What is a function pointer? How is it used?
6. Explain the difference between static and dynamic memory allocation.
7. What are the advantages of linked lists over arrays?
8. Describe the difference between singly, doubly, and circular linked lists.
9. What is a stack and queue? How are they implemented using arrays and linked lists?
10. Explain Big-O notation and its significance in algorithm analysis.
11. What are hash tables? How do they work?
12. Compare Merge Sort and Quick Sort in terms of time complexity and stability.
13. What is binary search? When is it most efficient?
14. How does dynamic programming differ from recursion?
15. Explain the difference between pass by value and pass by reference.
16. What is a memory leak? How can it be avoided?
17. How does garbage collection work in C? (Hint: Manual vs. Automatic)
18. What is a self-referential structure? Give an example.
19. Explain the typedef keyword and its use cases.
20. What is endianness? How does it affect data storage?
21. Explain the const keyword and its different use cases.
22. What is pointer arithmetic? Provide an example.
23. How does variable argument lists (va_list) work in C?
24. What is the difference between deep copy and shallow copy?
25. Explain memory alignment and padding in structures.
26. What is a circular buffer? Where is it used?
27. How does Dijkstra's algorithm work?
28. What is a trie (prefix tree)? Explain its applications.
29. What are B-trees and B+ trees? How are they used in databases?
30. Explain AVL trees and their balancing mechanisms.
31. What is Red-Black Tree? How does it differ from AVL?
32. Explain graph representations (adjacency matrix vs. adjacency list).
33. What is topological sorting? Where is it used?
34. Explain Kruskal's and Prim's algorithms for MST.
35. What is dynamic memory fragmentation? How can it be minimized?
36. Explain LRU (Least Recently Used) cache implementation.

37. What is tail recursion? How is it optimized?
38. Explain inline functions vs. macros.
39. What is Duff's device? How does it optimize loops?
40. Explain function overloading in C (using _Generic).
41. What is restrict keyword in C?
42. How does qsort() work in C?
43. What is Combinatorial Game Theory in algorithms?
44. Explain NP-complete problems with examples.
45. What is memoization? How does it optimize recursion?
46. Explain greedy algorithms vs. dynamic programming.
47. What is backtracking? Provide an example (e.g., N-Queens).
48. Explain bit manipulation tricks (e.g., counting set bits).
49. What is segmentation fault? How to debug it?

Coding Part

1. Reverse a string in-place.
2. Check if a string is a palindrome.
3. Implement strcpy(), strcat(), strcmp().
4. Find the first non-repeating character in a string.
5. Remove duplicates from a sorted array.
6. Implement a linked list with insert, delete, and reverse operations.
7. Print "Hello, World!" N times without loops (Recursion)
8. Check if a number is Armstrong (e.g., 153 = $1^3 + 5^3 + 3^3$)
9. Find the GCD of two numbers using recursion (Euclidean algorithm)
10. Calculate factorial without recursion (Iterative approach)
11. Print Fibonacci series up to N terms using just two variables
12. Check if a number is a palindrome (Reverse and compare)
13. Count vowels and consonants in a string (Case-insensitive)
14. Convert a decimal number to binary without arrays (Bitwise ops)
15. Find the sum of all digits until a single digit remains (Digital root)
16. Remove all whitespace from a string in-place
17. Check if two strings are anagrams (O(n) time, no sorting)
18. Count the occurrences of a substring in a string (Without strstr())
19. Reverse words in a sentence (e.g., "Hello World" → "World Hello")
20. Find the missing number in an array of 1 to N (XOR trick)
21. Separate even and odd numbers in an array (In-place, two- pointer)
22. Find all triplets in an array that sum to zero ($O(n^2)$ time)
23. Rotate an array left by D positions (Reversal algorithm, O(1) space)
24. Merge two sorted arrays into a third sorted array (O(m+n) time)
25. Implement strlen(), strcpy(), strcmp() without lib functions
26. Reverse an array using pointers
27. Find the largest element in an array using pointers
28. Concatenate two strings using pointers
29. Swap two arrays using pointers
30. Find the longest word in a string
31. Swap two numbers using pointers
32. Find the odd numbers in an array
33. Find the even numbers in an array

34. Detect a cycle in a linked list (Floyd's Algorithm).
35. Merge two sorted linked lists.
36. Sort a linked list using merge sort
37. Find the middle of a linked list in one pass.
38. Reverse a linked list in groups of k.
39. Print "Hello, World!" without using a semicolon (;).
40. Find the sum of digits of a number recursively.
41. Check if a number is even or odd without using % or /. (Hint: Use bitwise operations)
42. Print the binary representation of a number.
43. Swap two variables using XOR (no temporary variable).
44. Check if a number is a power of 2.
45. Count the number of 1s (set bits) in an integer. (Bit manipulation)
46. Reverse the bits of a given integer.
47. Implement isPrime() without using loops (recursion only).
48. Print a pyramid pattern using recursion.
49. Implement a stack using an array.
50. Implement a queue using linked lists.
51. Find the majority element in an array (appears > n/2 times). (Boyer-Moore Voting Algorithm)
52. Rotate a 2D matrix by 90 degrees in-place.
53. Find the smallest missing positive integer in an unsorted array. (O(n) time, O(1) space)
54. Remove all occurrences of a substring from a string in-place.
55. Find the longest substring with at most k distinct characters. (Sliding Window)
56. Clone a linked list with random pointers. (O(n) time)
57. Add two numbers represented as linked lists (MSB first).
58. Rearrange a linked list in zig-zag fashion (a < b > c < d > e ...).
59. Flatten a multilevel doubly linked list (DFS-style).
60. Detect and remove the longest palindrome list in a linked list.
61. Print the boundary traversal of a binary tree. (Anti- clockwise)
62. Convert a binary tree into a circular doubly linked list. (In-place)
63. Find the largest BST subtree in a binary tree.
64. Count the number of ways to decode a message (A=1, B=2, ..., Z=26).
65. Find the longest palindromic subsequence (LPS).
66. Egg Dropping Problem (min trials to find critical floor).
67. Maximum profit in a grid with obstacles (Robot Path).
68. Word Break Problem (check if a string can be segmented).
69. Serialize and deserialize an N-ary tree.
70. Check if two binary trees are mirror images.
71. Check for balanced parentheses using a stack.
72. Implement Binary Search recursively and iteratively.
73. Find the kth smallest element in an unsorted array.
74. Implement Bubble Sort, Selection Sort, Insertion Sort.
75. Implement Quick Sort with partitioning.
76. Implement Merge Sort with recursion.
77. Find all pairs in an array that sum to a given value.
78. Rotate an array by n positions.
79. Implement a Binary Search Tree (BST) with insertion, deletion, and traversal.
80. Find the LCA (Lowest Common Ancestor) in a BST.
81. Check if a binary tree is a BST.
82. Implement BFS and DFS for a graph.

83. Detect a cycle in a directed graph.
84. Implement Dijkstra's Algorithm.
85. Implement a priority queue using a heap.
86. Implement Trie for dictionary operations.
87. Count the number of set bits in an integer.
88. Swap two numbers without a temporary variable.
89. Reverse bits of a number.
90. Implement memcpy() and memset().
91. Implement atoi() and itoa().
92. Find the factorial of a number using recursion.
93. Print Fibonacci series iteratively and recursively.
94. Solve the Tower of Hanoi problem.
95. Implement N-Queens problem using backtracking.
96. Implement Knapsack Problem (0/1 and fractional).
97. Find the longest substring without repeating characters.
98. Implement LRU Cache using a hashmap and doubly linked list.
99. Read a file and count word frequencies.
100. Write a program to copy a file.
101. Implement a basic shell with fork() and exec().
102. Create a Makefile for a multi-file C project.
103. Write a program to list all files in a directory.
104. Implement a thread-safe queue.
105. Simulate producer-consumer problem using mutexes.
106. Implement a memory pool allocator.
107. Write a program to handle signals (SIGINT, SIGTERM).
108. Implement a simple HTTP server using sockets.
109. Check if a string is a palindrome.
110. Find the factorial of a number using recursion.
111. Swap two numbers without a temporary variable.
112. Implement a stack using an array.
113. Implement a queue using a linked list.
114. Reverse a linked list (iterative and recursive).
115. Detect a cycle in a linked list (Floyd’s algorithm).
116. Merge two sorted linked lists.
117. Find the middle element of a linked list in one pass.
118. Implement a binary search tree (BST) with insertion and traversal.
119. Check if a binary tree is balanced.
120. Find the height of a binary tree.
121. Sort an array using Quick Sort.
122. Implement a hash table with collision handling (chaining).
123. Find the longest substring without repeating characters.
124. Implement Dijkstra’s shortest path algorithm.
125. Detect if two linked lists intersect.
126. Find the kth smallest element in a BST.
127. Serialize and deserialize a binary tree.
128. Implement LRU Cache.
129. Find all permutations of a string.
130. Count the number of islands in a matrix (DFS/BFS).
131. Implement a priority queue using a heap.

132. Check if a binary tree is a valid BST.
133. Find the lowest common ancestor (LCA) in a BST.
134. Implement a trie (prefix tree).
135. Find the maximum subarray sum (Kadane's algorithm).
136. Rotate a matrix by 90 degrees.
137. Implement a circular buffer.
138. Find the longest palindromic substring.
139. Implement strstr() (substring search).
140. Convert a binary tree to a doubly linked list.
141. Find the median of two sorted arrays.
142. Implement a thread-safe singleton in C.
143. Generate all subsets of a set (power set).
144. Implement a thread pool in C.
145. Check if a graph is bipartite.
146. Implement A pathfinding algorithm.
147. Find the shortest path in a maze (BFS).
148. Implement a memory allocator (malloc/free).
149. Simulate a CPU scheduler (Round Robin, SJF).
150. Implement a file compression algorithm (Huffman coding).
151. Find the longest increasing subsequence (LIS).
152. Implement a concurrent linked list with locks.
153. Solve the N-Queens problem using backtracking.
154. Implement a garbage collector in C.
155. Find the maximum XOR of two numbers in an array.
156. Implement a thread-safe queue.
157. Write a program to detect memory leaks.
158. Implement your own malloc() and free() using sbrk().
159. Write a program to simulate ls -l (file permissions, size, etc.).
160. Create a shared library in C and dynamically load it.
161. Write a minimal shell that supports pipes (|).
162. Simulate the cp command with memory-mapped files (mmap).

Part 2: Operating Systems

Advanced Processes & Threads

1. What is a process control block (PCB)? What information does it store?
2. Explain thread synchronization in multi-threaded programs.
3. What is a zombie process? How can it be avoided?
4. Compare user-level threads vs kernel-level threads.
5. What is a daemon process? Give examples.

CPU Scheduling

6. Explain Multilevel Queue Scheduling with an example.
7. What is convoy effect in FCFS scheduling?
8. How does Shortest Remaining Time First (SRTF) work?
9. What is CPU affinity? Why is it useful?
10. Explain Lottery Scheduling and its fairness.

Process Synchronization

11. What is the critical section problem?
12. Explain Peterson's Solution for mutual exclusion.
13. How do test-and-set and compare-and-swap (CAS) instructions work?
14. What is a monitor? How does it ensure synchronization?
15. Explain the dining philosophers problem and its solutions.

Deadlocks

16. What are the four necessary conditions for a deadlock?
17. Explain resource allocation graph (RAG) for deadlock detection.
18. What is deadlock avoidance vs deadlock prevention?
19. How does the Banker's Algorithm work?
20. What is priority inversion? How is it resolved?

Memory Management

21. Explain segmentation vs paging.
22. What is internal and external fragmentation?
23. How does virtual memory work?
24. Explain page table structures (Hierarchical, Hashed, Inverted).
25. What is TLB (Translation Lookaside Buffer)?

File Systems & Disk Management

26. Explain inode structure in UNIX file systems.
27. What is journaling in file systems?
28. Compare FAT, NTFS, and ext4 file systems.
29. Explain RAID levels (0, 1, 5, 10).
30. What is wear leveling in SSDs?

Advanced Concepts

31. What is copy-on-write (COW) in process creation?
32. Explain memory-mapped files (mmap).
33. What is IPC (Inter-Process Communication)? Compare pipes, shared memory, and message queues.
34. Explain asynchronous I/O vs synchronous I/O.
35. What is NUMA (Non-Uniform Memory Access)?

Real-Time & Distributed Systems

36. What is a real-time operating system (RTOS)?
37. Explain hard real-time vs soft real-time systems.
38. What is Byzantine fault tolerance?
39. Explain Lamport's logical clocks for distributed systems.

40. What is CAP theorem in distributed systems?

Security & Virtualization

41. What is ASLR (Address Space Layout Randomization)?
42. Explain buffer overflow attacks and prevention techniques.
43. What is sandboxing in OS security?
44. Compare Type-1 vs Type-2 hypervisors.
45. What is containerization (Docker vs VMs)?

Kernel & System Calls

46. Explain system call execution flow (User → Kernel mode transition).
47. What is a kernel panic? Common causes?
48. How does Linux Completely Fair Scheduler (CFS) work?
49. What is O(1) scheduler in Linux?
50. Explain tickless kernel in modern OS.

Coding Part

Process & Thread Management

1. Create a child process using fork() and print PID/PPID.
2. Implement process chain (parent → child → grandchild).
3. Write a program to list all running processes (using /proc).
4. Create multiple threads and synchronize using pthread_mutex.
5. Simulate race condition and fix it with mutex.

IPC (Inter-Process Communication)

6. Implement unnamed pipe communication between parent & child.
7. Create a named pipe (FIFO) for IPC.
8. Implement shared memory between two processes.
9. Simulate producer-consumer problem using semaphores.
10. Implement a message queue using msgget(), msgsnd(), msgrcv().

Synchronization & Deadlocks

11. Solve dining philosophers problem using mutexes.
12. Implement reader-writer problem with priority to writers.
13. Simulate Banker's Algorithm for deadlock avoidance.
14. Implement priority inversion and fix it with priority inheritance.
15. Write a spinlock implementation in C.

Memory Management

16. Simulate page replacement algorithms (FIFO, LRU, Optimal).
17. Implement malloc() and free() using linked lists.
18. Write a memory allocator with fixed-size blocks.
19. Simulate buddy system memory allocation.
20. Detect memory leaks using custom wrappers.

File Systems & I/O

21. Write a program to copy a file using system calls (open, read, write).
22. Implement file locking (flock() or fcntl()).
23. Simulate log-structured file system (LFS) operations.
24. Write a simple shell supporting ls, cd, pwd.
25. Implement file search utility (like find).

Networking & Sockets

26. Create a TCP echo server & client.

27. Implement a UDP chat application.
28. Simulate HTTP GET request using sockets.
29. Write a port scanner in C.
30. Implement concurrent server using fork().

Kernel & System Programming

31. Write a Linux kernel module that prints "Hello, Kernel!".
32. Implement a custom system call in Linux.
33. Write a loadable kernel module (LKM) for a character device.
34. Simulate interrupt handling in a kernel module.
35. Implement procfs entry to read/write kernel variables.

Real-Time & Embedded Systems

36. Simulate RTOS task scheduling (Rate Monotonic).
37. Implement priority-based scheduler in userspace.
38. Write a watchdog timer in C.
39. Simulate hard real-time constraints using clock_nanosleep().
40. Implement cyclic executive scheduler.

Security & Debugging

41. Write a program to detect buffer overflow vulnerabilities.
42. Implement ASLR bypass (for educational purposes).
43. Simulate privilege escalation using setuid().
44. Write a strace-like tool using ptrace().
45. Implement core dump analyzer for crash debugging.

Advanced Problems

46. Simulate virtual memory paging with MMU emulation.
47. Implement file system in userspace (FUSE).
48. Write a mini OS scheduler in C.
49. Simulate distributed consensus (Paxos/Raft).
50. Implement a simple hypervisor using KVM.

Part 3: Linux System Programming

System Basics

1. What happens when you execute ls -l in Linux? (Explain shell → kernel flow)
2. How do environment variables work in Linux? How are they inherited?
3. Explain the difference between hard links and symbolic links.
4. What is the significance of /proc and /sys filesystems?
5. How does Linux handle file permissions (rwx for user/group/others)?

Process Management

6. Explain the difference between fork(), vfork(), and clone().
7. What happens during exec() system call? Does it create a new process?
8. How does wait() and waitpid() work? What are zombie processes?
9. What is a session and process group in Linux?
10. Explain the role of init process (PID 1) in Linux.

Signals & Interrupts

11. What are Linux signals? List 5 common signals and their uses.
12. How does sigaction() differ from signal()?
13. What is the difference between masking and blocking signals?
14. Explain real-time signals (SIGRTMIN to SIGRTMAX).
15. How can you send a signal to another process programmatically?

IPC (Inter-Process Communication)

16. Compare pipes, FIFOs, and Unix domain sockets.
17. When would you use shared memory vs message queues?
18. Explain mmap() for file/device mapping.
19. What are POSIX semaphores vs System V semaphores?
20. How does ftok() generate a key for IPC mechanisms?

File & I/O Operations

21. Explain file descriptors vs FILE* streams.
22. What is the difference between O_SYNC and O_DIRECT flags in open()?
23. How does lseek() work for random file access?
24. What are inotify APIs used for?
25. Explain scatter-gather I/O using readv()/writev().

Memory Management

26. How does malloc() work in Linux? Does it always use brk()/sbrk()?
27. What is memory overcommit in Linux?
28. Explain madvise() and its performance impact.
29. What are huge pages? How are they configured?
30. How does mlock() prevent memory swapping?

Threads & Synchronization

31. Compare pthreads vs Linux clone() threads.
32. What is thread-local storage (TLS)? How is it implemented?
33. Explain pthread mutexes vs futexes.
34. How do read-write locks improve performance?
35. What is a thread pool? When is it useful?

Networking & Sockets

36. Explain the difference between stream and datagram sockets.
37. What is the role of SO_REUSEADDR socket option?
38. How does epoll() differ from select()/poll()?

39. What are Unix domain sockets? When are they faster than TCP?
40. Explain zero-copy I/O techniques like splice().

Advanced Topics

41. What is seccomp? How does it restrict system calls?
42. Explain capabilities in Linux (e.g., CAP_NET_ADMIN).
43. How does ptrace() work for debugging/stracing?
44. What is cgroups and how does it limit resources?
45. Explain eBPF and its use cases in Linux.

Kernel Interaction

46. How do ioctl() calls communicate with device drivers?
47. What is sysfs and how is it used for device management?
48. Explain netlink sockets for kernel-userspace communication.
49. How are system calls implemented in Linux (from glibc to kernel)?
50. What is VDSO and how does it optimize system calls?

Coding Part

File & I/O Operations

1. Implement cat command to display file contents.
2. Write a program to copy files using read()/write().
3. Create a program that appends text to a file atomically (using O_APPEND).
4. Implement tail -f functionality using inotify.
5. Write a program to search for a string in files (like grep).

Process Management

6. Create a process tree (parent → child → grandchild) and print PIDs.
7. Implement a shell that runs commands with fork() + exec().
8. Write a program to measure process execution time using times().
9. Simulate nohup to detach a process from terminal.
10. Create a daemon process (detach from terminal, fork twice).

Signals

11. Write a signal handler to gracefully shutdown on SIGINT.
12. Implement a program that blocks SIGTERM but allows SIGKILL.
13. Create a SIGCHLD handler to reap zombie processes.
14. Use sigprocmask() to block signals during critical sections.
15. Write a program that sends signals between processes using kill().

IPC (Pipes, FIFOs, Shared Memory)

16. Implement pipe communication between parent and child processes.
17. Create a chat program using FIFOs (named pipes).
18. Use mmap() to share memory between two processes.
19. Implement a producer-consumer system using System V shared memory.
20. Write a program to pass file descriptors between processes using sendmsg().

Threads & Synchronization

21. Create two threads that increment a shared counter (with/without mutex).
22. Implement a thread-safe queue using pthread_mutex.
23. Solve the reader-writer problem with priority to writers.
24. Use pthread_barrier to synchronize multiple threads.
25. Write a program to deadlock two threads and then resolve it.

Sockets & Networking

26. Implement a TCP echo server and client.
27. Create a UDP-based file transfer program.

28. Write a concurrent server using fork() for multiple clients.
29. Use epoll() to handle 10K+ connections efficiently.
30. Implement HTTP GET request parsing in a server.

Memory & Performance

31. Write a custom malloc() using sbrk().
32. Allocate memory aligned to 64 bytes using posix_memalign().
33. Use madvise() to optimize memory access patterns.
34. Implement a memory leak detector using LD_PRELOAD.
35. Write a program to demonstrate copy-on-write with fork().

Advanced System Programming

36. Create a ptrace()-based debugger to trace system calls.
37. Implement a strace-like tool using ptrace().
38. Write a program to list open files of a process (/proc/<pid>/fd).
39. Use ioctl() to fetch terminal size (TIOCGWINSZ).
40. Simulate lsmod to list kernel modules.

Kernel Interaction

41. Write a netlink-based userspace-kernel communication program.
42. Create a sysfs entry to read/write kernel variables.
43. Use perf_event_open() to monitor CPU cache misses.
44. Implement a basic eBPF program to trace system calls.
45. Write a program to manipulate cgroups for CPU limiting.

Security & Real-World

46. Drop root privileges permanently using setuid().
47. Implement a chroot() jail for process isolation.
48. Write a seccomp filter to block execve().
49. Use capabilities to allow a non-root process to bind to port 80.
50. Create a program that detects buffer overflow attacks.

Part 4: Embedded Systems & ARM Architecture

Embedded Fundamentals

1. What defines an embedded system? How does it differ from general computing?
2. Explain the typical embedded system design workflow (from requirements to deployment)
3. Compare bare-metal programming vs RTOS-based development
4. What are the key constraints in embedded systems? (Power, Memory, Real-time)
5. Explain the role of watchdog timers in embedded systems

ARM Architecture

6. Compare ARM Cortex-M, Cortex-R, and Cortex-A series processors
7. Explain the ARM 3-stage and 5-stage pipeline architectures
8. What are the key differences between ARM and RISC-V architectures?
9. Describe the ARM register set (R0-R15, CPSR)
10. What are the various ARM processor modes? (User, IRQ, FIQ, Supervisor etc.)

Memory Systems

11. Explain Harvard vs Von Neumann architectures in ARM MCUs
12. What are the different memory types in embedded systems? (Flash, SRAM, EEPROM)
13. How does memory-mapped I/O work in ARM systems?
14. Explain the concept of bit-banding in ARM Cortex-M
15. What is Tightly Coupled Memory (TCM) in ARM processors?

Interrupts & Exceptions

16. Explain the ARM exception handling process
17. What's the difference between IRQ and FIQ in ARM?
18. How does nested interrupt handling work in ARM?
19. Explain the NVIC (Nested Vectored Interrupt Controller) in Cortex-M
20. What are the various ARM exception types? (Reset, NMI, HardFault etc.)

Power Management

21. Explain different low-power modes in ARM processors
22. How does the WFI (Wait For Interrupt) instruction work?
23. What are the techniques for power optimization in embedded designs?
24. Explain dynamic voltage and frequency scaling (DVFS) in ARM SoCs
25. How does clock gating help in power reduction?

Peripheral Interfaces

26. Compare UART, SPI, and I2C protocols
27. Explain DMA operation in ARM-based systems
28. What are the key considerations for ADC interfacing?
29. How does PWM generation work in ARM timers?
30. Explain the working of ARM's General Purpose Timer

Development & Debugging

31. What is the role of a JTAG debugger in embedded development?
32. Explain the ARM CoreSight debugging architecture
33. What are semihosting operations? When are they used?
34. How does SWD (Serial Wire Debug) differ from JTAG?
35. Explain the role of bootloaders in ARM systems

Advanced Concepts

36. What is TrustZone technology in ARM processors?
37. Explain the MPU (Memory Protection Unit) in ARM Cortex-M
38. How does cache coherency work in multi-core ARM systems?

39. What are the security considerations in ARM-based IoT devices?
40. Explain ARM's AMBA (Advanced Microcontroller Bus Architecture)

RTOS Considerations

41. How does context switching work in ARM for RTOS?
42. What are the key differences between FreeRTOS and Zephyr for ARM?
43. Explain priority inversion and its solutions in ARM RTOS
44. How are mutexes and semaphores implemented at the ARM assembly level?
45. What is the role of the SysTick timer in RTOS scheduling?

Optimization Techniques

46. Explain ARM NEON technology and its applications
47. What are the benefits of ARM's Thumb-2 instruction set?
48. How to optimize C code for ARM architectures?
49. Explain the use of ARM intrinsic functions
50. What are the key considerations for writing interrupt-safe code on ARM?

Coding Part:

ARM Assembly Fundamentals

1. Write ARM assembly to add two 64-bit numbers
2. Implement a delay loop using ARM assembly
3. Create an assembly function to enable IRQ interrupts
4. Write assembly code to switch from User to Supervisor mode
5. Implement memory copy using ARM assembly (with/without NEON)

Register & Bit Manipulation

6. Set/Clear/Toggle specific bits in a GPIO register
7. Implement a bit-banged SPI master in C
8. Write code to configure alternate function modes for GPIO pins
9. Create a circular buffer using bit masking operations
10. Implement a software debounce for button inputs

Interrupt Handling

11. Set up an external interrupt on GPIO pin
12. Implement a UART receive interrupt handler
13. Create a SysTick timer interrupt for periodic tasks
14. Write a nested interrupt handler with priority management
15. Implement a software interrupt (SWI) handler

Peripheral Drivers

16. Write a UART driver with polling and interrupt modes
17. Implement an I2C master driver
18. Create a PWM driver with variable duty cycle
19. Develop an ADC driver with DMA support
20. Write a driver for external flash memory (SPI interface)

Memory Management

21. Implement a memory allocator for embedded systems
22. Write code to relocate vector table in SRAM
23. Create a memory test pattern generator
24. Implement ECC (Error Correcting Code) for flash memory
25. Write linker script for custom memory layout

Power Management

26. Implement entry/exit from low-power sleep mode
27. Write code for dynamic clock scaling

28. Create a battery monitoring system
29. Implement a watchdog timer with refresh logic
30. Write power measurement code using current sensing

RTOS Integration

31. Create FreeRTOS tasks for sensor sampling
32. Implement a message queue between RTOS tasks
33. Write a memory pool allocator for RTOS
34. Create a priority inheritance mutex implementation
35. Develop a software timer management system

ARM Optimization

36. Optimize a FIR filter using ARM DSP instructions
37. Implement memcpy with NEON intrinsics
38. Write cycle-accurate delay functions
39. Create a CRC32 calculation using ARM instructions
40. Optimize floating-point operations on Cortex-M4

Debugging & Testing

41. Implement a debug log over UART
42. Write a memory corruption detector
43. Create a CPU usage monitor
44. Implement a hardware exception handler
45. Write a test harness for peripheral validation

Advanced Projects

46. Develop a bootloader with firmware update capability
47. Implement a simple filesystem for flash memory
48. Create a command-line interface over UART
49. Write a power-fail safe data logging system
50. Develop a BLE (Bluetooth Low Energy) peripheral

Key Focus Areas in Problems

- Hardware Awareness: Direct register manipulation
- Real-Time Constraints: Deadline meeting in ISRs
- Resource Efficiency: Minimal memory/CPU usage
- Reliability: Watchdog, error recovery
- Low-Power Operation: Sleep mode transitions

Recommended Tools

- Compilers: ARM GCC, Keil, IAR
- Debuggers: J-Link, ST-Link, OpenOCD
- Boards: STM32 Discovery, NXP FRDM, Raspberry Pi Pico
- RTOS: FreeRTOS, Zephyr, Mbed OS

Part 5: Kernel & Device Drivers

Kernel Fundamentals

1. Compare monolithic, microkernel, and hybrid kernel architectures.
2. What is the role of the system call table in Linux?
3. Explain the kernel space vs user space separation.
4. What are Loadable Kernel Modules (LKMs)? How are they different from built-in drivers?
5. Describe the Linux kernel boot process from BIOS to init.

Process & Memory Management

6. How does the kernel manage process descriptors (task_struct)?
7. Explain virtual memory management in Linux (vm_area_struct, page tables).
8. What is Direct Memory Access (DMA)? How does the kernel handle it?
9. Describe kernel memory allocators (kmalloc, vmalloc, slab allocator).
10. What is memory-mapped I/O (MMIO) vs port-mapped I/O (PMIO)?

Synchronization & Concurrency

11. Why is synchronization critical in kernel programming?
12. Compare spinlocks, mutexes, and semaphores in the kernel.
13. What is RCU (Read-Copy-Update)? When is it preferred?
14. Explain deadlock scenarios in kernel drivers.
15. What is priority inversion and how does the kernel prevent it?

Interrupts & Bottom Halves

16. How does the kernel handle hardware interrupts (IRQs)?
17. Explain the difference between top halves and bottom halves in interrupt handling.
18. What are tasklets, softirqs, and workqueues?
19. How does threaded IRQ handling improve latency?
20. What is interrupt coalescing?

Device Drivers

21. What is the Linux Device Model (kobject, kset, sysfs)?
22. Explain the probe() and remove() functions in device drivers.
23. How are character devices different from block devices?
24. What is the role of file_operations in Linux drivers?
25. Describe platform devices and device tree bindings.

File Systems & Block I/O

26. How does the VFS (Virtual File System) layer work?
27. Explain the bio layer in block device drivers.
28. What is request merging in the I/O scheduler?
29. Compare ext4, Btrfs, and XFS file systems.
30. How does FUSE (Filesystem in Userspace) work?

Networking & PCI

31. Explain the network device driver architecture (net_device).
32. What is NAPI (New API) for network drivers?
33. How does PCI/PCIe device enumeration work in Linux?
34. Describe USB driver architecture (usb_driver, urb).
35. What is DMA-BUF for zero-copy buffer sharing?

Debugging & Profiling

36. How do you debug a kernel crash (Oops, panic)?
37. Explain ftrace, kprobes, and perf for kernel tracing.
38. What is KASAN (Kernel Address Sanitizer)?

39. How does KGDB (Kernel GNU Debugger) work?
40. What are kernel livepatching techniques?

Security & Real-World Considerations

41. How does SELinux enforce security in the kernel?
42. What are kernel hardening techniques (CONFIG_STACKPROTECTOR)?
43. Explain secure boot and signed kernel modules.
44. How does Control Groups (cgroups) limit resource usage?
45. What is Kernel Samepage Merging (KSM)?

Advanced Topics

46. How do eBPF (Extended Berkeley Packet Filter) programs work?
47. Explain asymmetric multi-processing (AMP) in Linux.
48. What is real-time Linux (PREEMPT_RT)?
49. Describe virtualization in Linux (KVM, containers).
50. How does ARM TrustZone integrate with Linux?

Coding Part:

Basic Kernel Modules

1. Write a "Hello, Kernel!" module that logs to dmesg.
2. Create a module that lists all running processes (for_each_process).
3. Implement a module that reads/writes /proc entries.
4. Write a module that creates a character device (mknod).
5. Develop a module that uses kernel timers (timer_list).

Memory Management

6. Allocate contiguous memory with kmalloc and dma_alloc_coherent.
7. Implement memory mapping (mmap) in a character driver.
8. Write a slab cache allocator for custom objects.
9. Simulate a memory leak and detect it with kmemleak.
10. Use vmalloc to allocate large non-contiguous memory.

Synchronization

11. Implement a mutex to protect shared data in a driver.
12. Write a spinlock-based atomic counter.
13. Use RCU for read-mostly data structures.
14. Simulate a deadlock between two kernel threads.
15. Fix priority inversion with priority inheritance mutexes.

Interrupt Handling

16. Write a driver for a GPIO interrupt (e.g., button press).
17. Implement IRQ sharing between multiple devices.
18. Use tasklets to defer interrupt processing.
19. Develop a threaded IRQ handler for high-latency devices.
20. Simulate interrupt throttling to reduce CPU load.

Block & Network Drivers

21. Write a RAM disk block driver (register_blkdev).
22. Implement I/O scheduling (elevator API) for a block device.
23. Develop a null network driver (net_device_ops).
24. Simulate packet filtering with netfilter hooks.
25. Write a USB HID driver for a custom device.

File Systems

26. Create a pseudo-filesystem (procfs, sysfs).
27. Implement a FUSE-based encrypted filesystem.
28. Write a loop device driver for file-backed storage.
29. Develop a logger that appends to a file from kernel space.
30. Simulate file permission checks (inode_permission).

Debugging & Profiling

31. Use printk with log levels (KERN_DEBUG, KERN_ERR).
32. Write a kernel panic handler (panic_notifier).
33. Trace function calls with ftrace.
34. Use kprobes to hook into kernel functions.
35. Profile CPU usage with perf events.

Advanced Drivers

36. Develop a PCIe driver for a custom FPGA device.
37. Write a DMA engine driver for scatter-gather transfers.
38. Implement userspace I/O (UIO) for hardware access.
39. Create a virtual sensor driver (IIO subsystem).
40. Simulate a battery/power management driver.

Security & Real-World

41. Sign a kernel module with OpenSSL.
42. Use SELinux hooks to restrict device access.
43. Implement secure memory wiping (memset_secure).
44. Write a kernel firewall with netfilter.
45. Simulate DMA attacks and mitigation techniques.

Performance & Optimization

46. Optimize a driver with inline assembly (asm volatile).
47. Use SIMD (NEON/SSE) in kernel code.
48. Implement zero-copy networking (splice, sendfile).
49. Write a multi-queue block driver for SSDs.
50. Benchmark context switch latency (cyclictest).

Key Tools & Techniques

- Debugging: printk, gdb, kgdb, kdump
- Tracing: ftrace, perf, eBPF, LTTng
- Testing: kunit, kselftest, QEMU for virtual hardware
- Performance: perf stat, vmstat, iostat

Part 6: Networking & TCP/IP Stack

Protocol Fundamentals

1. Explain the OSI 7-layer model vs TCP/IP 4-layer model
2. How does Ethernet framing work (MAC addresses, VLAN tagging)?
3. What is the difference between connection-oriented (TCP) and connectionless (UDP) protocols?
4. Explain IP fragmentation and MTU/MSS concepts
5. How does ARP resolve IP addresses to MAC addresses?

IP Layer

6. Compare IPv4 and IPv6 header structures
7. Explain subnetting and CIDR notation
8. What is NAT (Network Address Translation)? Types (SNAT, DNAT, PAT)?
9. How do routing protocols (RIP, OSPF, BGP) work?
10. Explain ICMP and its uses (ping, traceroute)

Transport Layer

11. Describe the TCP 3-way handshake and 4-way termination
12. What is TCP congestion control (Tahoe, Reno, CUBIC)?
13. Explain TCP flow control (sliding window, RWND)
14. How does UDP checksum work compared to TCP?
15. What are TCP options (MSS, SACK, Timestamps)?

Application Layer

16. Compare HTTP/1.1, HTTP/2, and HTTP/3
17. Explain DNS resolution (iterative vs recursive queries)
18. How does TLS handshake work (RSA vs ECDHE)?
19. Describe SMTP email delivery process
20. What is WebSocket and how does it differ from HTTP?

Network Programming

21. Explain socket API (socket(), bind(), listen(), accept())
22. What is the difference between select(), poll(), and epoll()?
23. How do non-blocking sockets work with EAGAIN/EWOULDBLOCK?
24. Describe zero-copy networking techniques (sendfile, splice)
25. What are Unix domain sockets and when to use them?

Kernel Networking

26. Explain the Linux network stack (from NIC to socket)
27. What is NAPI in Linux network drivers?
28. How does netfilter/iptables work (tables, chains)?
29. Describe TC (Traffic Control) and QoS in Linux
30. What are XDP (eXpress Data Path) and AF_XDP?

Advanced Topics

31. Explain QUIC protocol and its advantages
32. How does MPTCP (Multipath TCP) work?
33. Describe IPSec (AH vs ESP, transport/tunnel modes)
34. What is SDN (Software Defined Networking)?
35. Explain TCP/IP offloading (TOE, LRO, GRO)

Security

36. Compare stateful and stateless firewalls
37. How do SYN floods and DDoS attacks work?
38. Explain TLS 1.3 improvements over TLS 1.2

39. What is DNSSEC and how does it prevent spoofing?
40. Describe VPN technologies (IPSec, OpenVPN, WireGuard)

Wireless & IoT

41. Compare Wi-Fi 6 vs 5G technologies
42. Explain BLE (Bluetooth Low Energy) protocol stack
43. How does Zigbee mesh networking work?
44. What is LoRaWAN and its use cases?
45. Describe MQTT protocol for IoT communications

Performance

46. How to measure network latency vs throughput?
47. Explain TCP BBR congestion control algorithm
48. What causes bufferbloat and how to mitigate it?
49. How does kernel bypass (DPDK, RDMA) work?
50. Optimize HTTP/2 server push strategies

Coding Part:

Socket Programming

1. Implement TCP echo server/client
2. Create UDP broadcast/multicast sender/receiver
3. Build HTTP 1.0 server (GET/POST handling)
4. Write non-blocking TCP chat server using select()
5. Implement proxy server (forward TCP connections)

Protocol Implementation

6. Simulate ARP cache with timeout
7. Implement ICMP ping (raw sockets)
8. Build DNS resolver (UDP queries)
9. Write DHCP client (discover/request)
10. Create TLS 1.2 handshake simulation

Kernel Networking

11. Develop netfilter module to log packets
12. Write XDP program to count packets
13. Implement TC classifier for QoS
14. Create virtual network device driver
15. Build eBPF program to monitor connections

Performance & Debugging

16. Measure TCP throughput between hosts
17. Implement packet capture (like tcpdump)
18. Write latency measurement tool
19. Create bandwidth throttler
20. Build packet reordering detector

Advanced Projects

21. Implement QUIC client over UDP
22. Create VPN tunnel using TUN/TAP
23. Write Tor-like onion routing prototype
24. Build SDN controller (OpenFlow)
25. Develop Wireshark dissector plugin

Part 7: RTOS & Real-Time Systems

RTOS Fundamentals

1. Compare RTOS vs GPOS design philosophies
2. Explain hard vs soft real-time requirements
3. What is determinism in RTOS contexts?
4. Describe priority inversion and solutions
5. Compare preemptive vs cooperative scheduling

Scheduling

6. Explain rate monotonic scheduling (RMS)
7. How does earliest deadline first (EDF) work?
8. Compare fixed-priority vs dynamic-priority schedulers
9. What is context switching overhead?
10. Describe tickless scheduling in RTOS

Memory Management

11. RTOS memory allocation strategies (pools, slabs)
12. How to avoid heap fragmentation in RTOS?
13. Explain MPU (Memory Protection Unit) usage
14. Compare static vs dynamic memory in RTOS
15. What is stack overflow protection?

IPC & Synchronization

16. RTOS message queues implementation
17. Compare mutexes vs binary semaphores
18. Explain priority inheritance protocol
19. How do mailboxes differ from queues?
20. Describe event flags pattern

Performance & Latency

21. Measure interrupt latency in RTOS
22. Explain WCET (Worst-Case Execution Time)
23. What causes jitter in real-time systems?
24. How to benchmark RTOS performance?
25. Describe cache-aware scheduling

RTOS Implementations

26. Compare FreeRTOS, Zephyr, and VxWorks
27. Explain FreeRTOS task states
28. How does RT-Thread IPC work?
29. Describe QNX microkernel architecture
30. What is Mbed OS scheduling model?

Advanced Topics

31. Explain mixed-criticality systems
32. How does AMP (Asymmetric MP) work in RTOS?
33. Describe TEE (Trusted Execution Environment)
34. What is DO-178C certification for avionics?
35. Explain time-triggered architecture

Coding Part:

Task Management

1. Create periodic tasks in FreeRTOS
2. Implement dynamic priority change
3. Write task watchdog monitor
4. Build task statistics collector
5. Develop task tracing system

Synchronization

6. Implement priority inheritance mutex
7. Solve dining philosophers problem
8. Write reader-writer lock
9. Create barrier synchronization
10. Develop event-driven state machine

Memory Management

11. Write memory pool allocator
12. Implement stack usage monitor
13. Create fixed-block allocator
14. Develop memory defragmentation
15. Write MPU region configurator

Performance

16. Measure context switch time
17. Implement WCET analyzer
18. Write interrupt latency test
19. Create scheduler stress test
20. Develop cache prefetching

Device Drivers

21. Write UART driver with DMA
22. Implement RTOS-aware SPI driver
23. Develop ADC sampling task
24. Create watchdog service
25. Write power management

Advanced Projects

26. Port FreeRTOS to RISC-V
27. Implement TLS 1.3 in RTOS
28. Build RTOS trace visualizer
29. Develop CAN bus stack
30. Create FAT filesystem

GOING ADVANCED

Part 1: C & Data Structures

1. How would you implement a lock-free linked list using CAS (Compare-And-Swap)?
2. Explain the ABA problem in lock-free algorithms and how to mitigate it.
3. Design a thread-safe, generic red-black tree in C using macros.
4. How does restrict keyword optimize pointer aliasing? Show disassembly examples.
5. Implement a memory allocator with O(1) allocation/deallocation using buddy system.
6. Write a C macro to simulate templated functions (e.g., MAX(T, x, y)).
7. Explain how to exploit undefined behavior (e.g., type-punning) for performance gains.
8. Design a concurrent hash table with striped locking and resizing.
9. How would you detect stack corruption using canary values?
10. Implement a coroutine scheduler in C using setjmp/longjmp.
11. Write a C interpreter in C (subset of C).
12. Implement a garbage collector for C (mark-and-sweep).
13. Build a minimal HTTP/1.1 server from scratch.
14. Write a bootloader in C (x86 assembly + C).
15. Create a minimal OS kernel with multitasking (context switching).

Part 2: Operating Systems

1. How does the Linux CFS (Completely Fair Scheduler) handle NUMA architectures?
2. Explain how eBPF hooks into the kernel to replace iptables for packet filtering.
3. Design a userspace page fault handler using userfaultfd().
4. How does the kernel mitigate Spectre/Meltdown vulnerabilities at runtime?
5. Implement a fault-tolerant filesystem using COW (Copy-On-Write) techniques.
6. Explain how io_uring achieves zero-copy I/O with kernel bypass.
7. How would you hotpatch a running kernel function using kprobes?
8. Design a deterministic memory allocator for real-time tasks.
9. Explain how KASLR (Kernel Address Space Layout Randomization) is bypassed in exploits.
10. Implement a minimal hypervisor using KVM APIs.

Part 3: Linux System Programming

1. How does seccomp enforce syscall filtering in containers (e.g., Docker)?
2. Design a shared library interposer to hijack malloc() calls.
3. Explain how vDSO accelerates syscalls like gettimeofday.
4. Implement a deadlock detector using ptrace and graph algorithms.
5. How would you mmap a file with fault injection for testing?
6. Design a userspace TCP stack using AF_PACKET raw sockets.
7. Explain how cgroups v2 isolates GPU resources.
8. Implement a crash-resistant logger using O_DIRECT and pwrite().
9. How does eBPF allow safe kernel scripting without modules?
10. Design a filesystem in userspace (FUSE) with encryption.

Part 4: Embedded Systems & ARM

1. How do you debug a hard fault on Cortex-M with no debugger?
2. Design a RTOS with memory protection using MPU and TrustZone.
3. Explain how to achieve deterministic interrupt latency on Cortex-R5.
4. Implement a secure bootloader with anti-rollback and measured boot.
5. How would you optimize an FFT for Cortex-M55 with Helium extensions?
6. Design a power-aware RTOS scheduler for battery-powered devices.
7. Explain how to use ARM ETM for real-time instruction tracing.
8. Implement a CAN FD driver with zero-copy DMA and hardware timestamps.
9. How do you mitigate Rowhammer attacks on LPDDR4 in automotive SoCs?
10. Design a fault-tolerant system using dual-core lockstep (Cortex-R52).

Part 5: Kernel & Device Drivers

1. How does the kernel enforce DMA coherency on ARM64 with non- cacheable mappings?
2. Design a PCIe driver with MSI-X interrupts and NUMA awareness.
3. Explain how io_uring bypasses the block layer for NVMe SSDs.
4. Implement a BPF-based network driver for DPDK-like performance.
5. How would you debug a kernel deadlock involving RCU and spinlocks?
6. Design a thermal governor with machine learning-based throttling.
7. Explain how KASAN detects out-of-bounds accesses in slab allocations.
8. Implement a live kernel patching framework for critical security fixes.
9. How does the kernel handle page faults for device memory (e.g., GPU VRAM)?
10. Design a filesystem with inline encryption (fscrypt) for eMMC.

Part 6: Networking & TCP/IP Stack

1. How does QUIC (HTTP/3) avoid head-of-line blocking at the transport layer?
2. Design a TCP congestion controller using reinforcement learning.
3. Explain how XDP (eXpress Data Path) achieves 100Gbps packet filtering.
4. Implement a userspace TLS 1.3 stack with kernel TLS offload.
5. How would you optimize the Linux kernel for 10µs RPC latency?
6. Design a SDN switch using P4 and programmable NICs.
7. Explain how BBR congestion control outperforms CUBIC in high-BDP networks.
8. Implement a zero-copy RDMA-based filesystem (e.g., NVMe-over- Fabrics).
9. How does TCP Fast Open (TFO) reduce HTTPS handshake latency?
10. Design a DDoS mitigation system using eBPF and hardware rate-limiting.

Part 7: RTOS & Real-time Systems

1. How do you achieve µs-level determinism in a multicore RTOS?
2. Design a mixed-criticality scheduler (e.g., ARM TrustZone + FreeRTOS).
3. Explain how to verify RTOS timing constraints with formal methods (TLA+).
4. Implement a memory-constrained RTOS with guaranteed WCET (Worst- Case Execution Time).
5. How would you port FreeRTOS to RISC-V with PMP (Physical Memory Protection)?
6. Design a RTOS for safety-critical systems (ISO 26262 ASIL-D compliant).
7. Explain how to use hardware timers for nanosecond-precision scheduling.
8. Implement a fault-tolerant RTOS with triple modular redundancy (TMR).
9. How do you debug priority inversion in a system with 100+ tasks?
10. Design an RTOS with support for probabilistic real-time tasks (e.g., AI inference).

Microcontrollers & Microprocessors

Raspberry Pi (Latest Models: Pi 5, Pi 4B, Pi Pico)

Raspberry Pi 5 (2023)

- **Processor:** Broadcom BCM2712 (4× Cortex-A76 @ 2.4GHz)
- **GPU:** VideoCore VII (OpenGL ES 3.1, Vulkan 1.2)
- **RAM:** 4GB/8GB LPDDR4X
- **Storage:** MicroSD, PCIe 2.0 x1 for NVMe SSD
- **GPIO:** 40-pin header (3.3V logic, 26× GPIO, UART, I2C, SPI, PWM)
- **USB:** 2× USB 3.0, 2× USB 2.0
- **Networking:** Gigabit Ethernet, Dual-band Wi-Fi 5, Bluetooth 5.0
- **Video Output:** 2× micro-HDMI (4K @ 60Hz) • **Power:** USB-C (5V/5A)
- **Peripherals:**
    - 2× MIPI CSI (camera) o 1× MIPI DSI (display)
    - o RTC (Real-Time Clock)

Raspberry Pi Pico (RP2040 MCU)

- **Processor:** Dual-core ARM Cortex-M0+ @ 133MHz
- **RAM:** 264KB SRAM
- **Flash:** 2MB QSPI
- **GPIO:** 26× multifunction (UART, I2C, SPI, PWM, ADC)
- **ADC:** 3× 12-bit (0-3.3V)
- **Interfaces:** USB 1.1 (Host/Device), PIO (Programmable I/O)

ESP32 (Espressif Systems)

ESP32-WROOM-32 (Common Variant)

- **Processor:** Dual-core Xtensa LX6 @ 240MHz
- **RAM:** 520KB SRAM (320KB for apps) • **Flash:** 4MB/16MB (SPI)
- **Wireless:**
    - Wi-Fi 4 (802.11 b/g/n) o Bluetooth 4.2 (BLE)
- **GPIO:** 34× (18× ADC, 2× DAC, 10× capacitive touch)
- **ADC:** 12-bit (0-3.3V) • **DAC:** 2× 8-bit
- **Interfaces:** o 3× UART o 2× I2C o 4× SPI o 16× PWM o CAN 2.0
- **Power:** 3.3V (100mA GPIO max)

ESP32-S3 (Latest)

- **Processor:** Dual-core Xtensa LX7 @ 240MHz
- **AI Acceleration:** Vector instructions for ML
- **RAM:** 512KB SRAM + 320KB ROM
- **GPIO:** 45× (USB OTG, LCD interface)

BeagleBone (BeagleBone Black, AI-64)

BeagleBone Black (BBB)

- **Processor:** TI AM3358 (1× Cortex-A8 @ 1GHz)
- **RAM:** 512MB DDR3
- **Storage:** 4GB eMMC, MicroSD
- **GPIO:** 92× (2× 46-pin headers, 65× usable) o 7× ADC (1.8V max) o 4× UART, 2× I2C, 2× SPI o 8× PWM
- **Networking:** 10/100 Ethernet
- **Power:** 5V DC (1A)

BeagleBone AI-64 (2022)

- **Processor:** TI AM625 (4× Cortex-A53 @ 1.4GHz + 2× Cortex-R5F)
- **GPU:** 2× Cortex-M4F, Imagination PowerVR GPU
- **RAM:** 4GB LPDDR4
- **Storage:** 16GB eMMC, MicroSD
- **GPIO:** 92× (1.8V/3.3V)
- **AI Acceleration:** 4 TOPS (TIDL)

Arduino Uno R3 (ATmega328P)

Specifications

- **Processor:** ATmega328P @ 16MHz (8-bit AVR)
- **RAM:** 2KB SRAM
- **Flash:** 32KB (0.5KB for bootloader)
- **EEPROM:** 1KB
- **GPIO:** 14× digital (6× PWM), 6× analog (10-bit ADC) • **Voltage Levels:** 5V logic
- **Interfaces:** o 1× UART o 1× I2C o 1× SPI
- **Power:** 7-12V DC (barrel jack) or 5V USB

STM32 (ARM Cortex-M Series)

STM32F103 (Blue Pill)

- **Processor:** Cortex-M3 @ 72MHz
- **RAM:** 20KB SRAM
- **Flash:** 64KB/128KB
- **GPIO:** 37× (5V-tolerant) • **ADC:** 2× 12-bit (16 channels)
- **Interfaces:** o 3× USART o 2× I2C o 2× SPI o 1× CAN

- **Power:** 3.3V (50mA per GPIO) **STM32H7 (High-Performance)**
- **Processor:** Cortex-M7 @ 480MHz + Cortex-M4 @ 240MHz
- **RAM:** 1MB SRAM (564KB DTCM)
- **Flash:** 2MB
- **GPIO:** 168× (3.3V) • **ADC:** 3× 16-bit
- **Interfaces:** o USB OTG HS/FS o Ethernet MAC o 4× I2C, 6× USART, 4× SPI

ARM Cortex-M Series (General Overview)

Core	Architecture	Max Clock	Features	Typical Use Cases
Cortex-M0	ARMv6-M	50 MHz	Ultra-low power, Thumb-2	Simple IoT, sensors
Cortex-M3	ARMv7-M	120 MHz	NVIC (Nested Vectored Interrupts), MPU	Industrial control, wearables
Cortex-M4	ARMv7E-M	240 MHz	DSP, FPU, single-cycle MAC	Audio processing, motor control
Cortex-M7	ARMv7E-M	480 MHz	Cache, TCM, dual-issue pipeline, FPU	HMI, edge AI, high-performance IoT
Cortex-M33	ARMv8-M	160 MHz	TrustZone, FPU, low-power modes	Secure IoT, low-power embedded
Cortex-M23	ARMv8-M	~50 MHz	TrustZone, ultra-low power	Secure ultra-low-power devices

✅ Notes:

- **Clock speed** varies by silicon vendor; values are common maximums from ARM-based MCUs.
- **Cortex-M23** added for completeness — it's the low-power TrustZone sibling of M33.
- Features like **MPU** = Memory Protection Unit, **TCM** = Tightly Coupled Memory, **DSP** = Digital Signal Processing.
- **Use cases** filled based on ARM documentation and industry examples.

ARM Microprocessors (Cortex-A Series)

Core	ISA	Cores	Max Clock	Typical Use Cases
Cortex-A53	ARMv8-A	1–8	2.0 GHz	Raspberry Pi 3, low-power Linux devices
Cortex-A72	ARMv8-A	2–8	2.5 GHz	Raspberry Pi 4, mid-range SBCs
Cortex-A76	ARMv8.2-A	1–8	3.0 GHz	High-performance single board computers
Cortex-X1	ARMv8.4-A	1–8	3.3 GHz	Flagship smartphones, high-end SoCs

✅ Notes:

- **ISA** = Instruction Set Architecture
- **SBC** = Single Board Computer
- All listed cores support 64-bit operation (AArch64)
- **Cores** column reflects typical scalable configurations
- Use cases derived from known implementations and public datasheets

Pinout Diagrams for Microcontrollers & Microprocessors

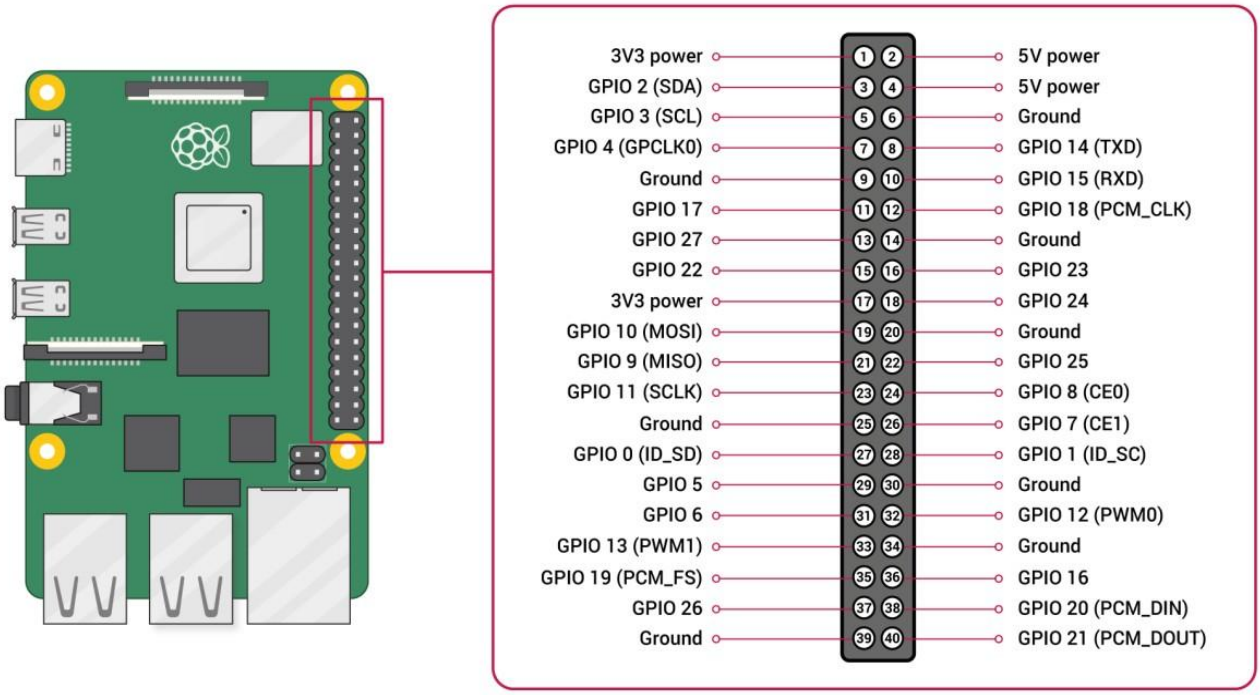
Raspberry Pi 5 (40-Pin Header)

Pinout:

3V3 (1) —— (2) 5V
GPIO2 (3) —— (4) 5V
GPIO3 (5) —— (6) GND
GPIO4 (7) —— (8) GPIO14 (TXD)
GND (9) —— (10) GPIO15 (RXD)
GPIO17 (11) —— (12) GPIO18 (PWM)
GPIO27 (13) —— (14) GND
GPIO22 (15) —— (16) GPIO23
3V3 (17) —— (18) GPIO24
GPIO10 (19) —— (20) GND
GPIO9 (21) —— (22) GPIO25
GPIO11 (23) —— (24) GPIO8
GND (25) —— (26) GPIO7
GPIO0 (27) —— (28) GPIO1
GPIO5 (29) —— (30) GND
GPIO6 (31) —— (32) GPIO12
GPIO13 (33) —— (34) GND
GPIO19 (35) —— (36) GPIO16
GPIO26 (37) —— (38) GPIO20
GND (39) —— (40) GPIO21

Key Pins:

- ✓ **GPIO2/3:** I2C1 (SDA/SCL)
- ✓ **GPIO14/15:** UART0 (TXD/RXD)
- ✓ **GPIO18:** Hardware PWM
- ✓ **GPIO10/11:** SPI0 (MOSI/MISO)
- ✓ **3V3/5V/GND:** Power rails


3V3 power 1 2 5V power
GPIO 2 (SDA) 3 4 5V power
GPIO 3 (SCL) 5 6 Ground
GPIO 4 (GPCLK0) 7 8 GPIO 14 (TXD)
Ground 9 10 GPIO 15 (RXD)
GPIO 17 11 12 GPIO 18 (PCM_CLK)
GPIO 27 13 14 Ground
GPIO 22 15 16 GPIO 23
3V3 power 17 18 GPIO 24
GPIO 10 (MOSI) 19 20 Ground
GPIO 9 (MISO) 21 22 GPIO 25
GPIO 11 (SCLK) 23 24 GPIO 8 (CE0)
Ground 25 26 GPIO 7 (CE1)
GPIO 0 (ID_SD) 27 28 GPIO 1 (ID_SC)
GPIO 5 29 30 Ground
GPIO 6 31 32 GPIO 12 (PWM0)
GPIO 13 (PWM1) 33 34 Ground
GPIO 19 (PCM_FS) 35 36 GPIO 16
GPIO 26 37 38 GPIO 20 (PCM_DIN)
Ground 39 40 GPIO 21 (PCM_DOUT)

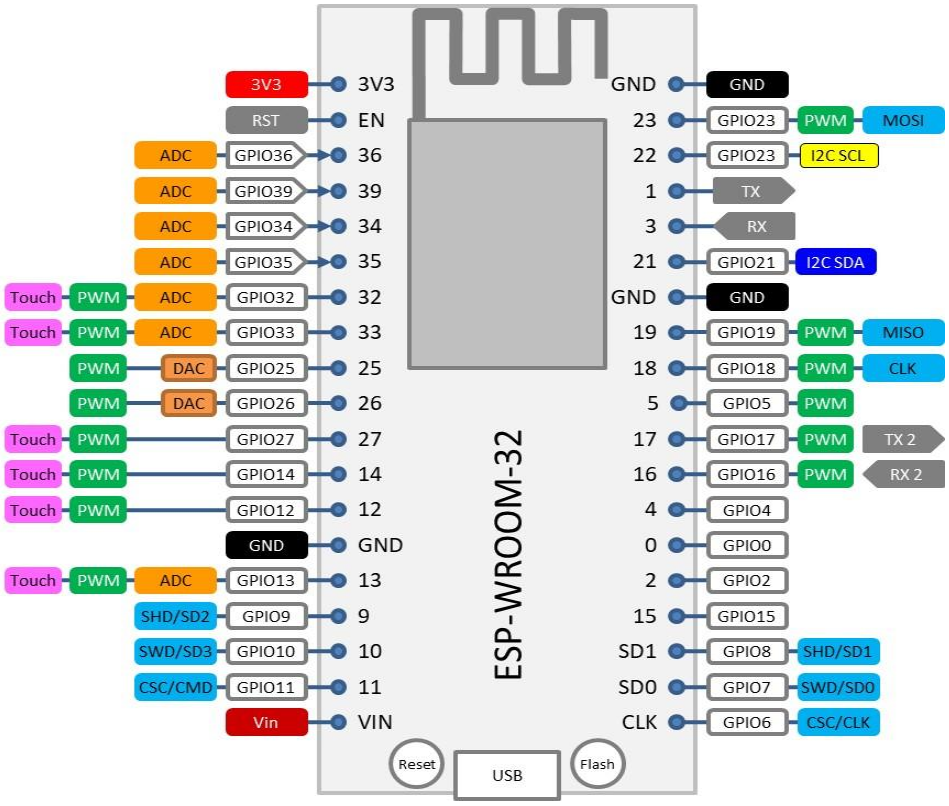
ESP32 (38-Pin DevKit)

Pinout:

3V3 —— EN
GND —— VP (GPIO36)
D15 —— VN (GPIO39)
D2 —— D34 (Input Only)
D4 —— D35 (Input Only)
D16 —— D32
D17 —— D33
D5 —— D25 (DAC1)
D18 —— D26 (DAC2)
D19 —— D27
D21 —— D14
RX2 —— D12
TX2 —— D13
D22 —— D23
D23 —— GND
5V —— 3V3

Key Pins:

- ✓ **3V3** → 3.3V output (for sensors, LEDs).
- ✓ **5V** → Input for USB/external power.
- ✓ **GND** → Ground (multiple pins).
- ✓ GPIO0 → LOW = Flashing mode, HIGH = Normal boot.
- ✓ EN → Reset (LOW = reset, HIGH = run).
- ✓ UART0 (GPIO1-TX, GPIO3-RX) → Debug serial (avoid using).
- ✓ I2C (GPIO21-SDA, GPIO22-SCL) → Default I2C bus.
- ✓ SPI (GPIO18-CLK, GPIO19-MISO, GPIO23-MOSI) → Default VSPI.
- ✓ GPIO36 (SVP), GPIO39 (SVN), GPIO34, GPIO35 → Input-only (no pull-ups).
- ✓ GPIO2, GPIO5, GPIO12, GPIO15 → Must be in correct state at boot.


3V3
3V3
RST
EN
ADC
GPIO36
36
ADC
GPIO39
39
ADC
GPIO34
34
ADC
GPIO35
35
Touch
PWM
ADC
GPIO32
32
Touch
PWM
ADC
GPIO33
33
PWM
DAC
GPIO25
25
PWM
DAC
GPIO26
26
Touch
PWM
GPIO27
27
Touch
PWM
GPIO14
14
Touch
PWM
GPIO12
12
GND
GND
Touch
PWM
ADC
GPIO13
13
SHD/SD2
GPIO9
9
SWD/SD3
GPIO10
10
CSC/CMD
GPIO11
11
Vin
VIN
GND
GND
23
GPIO23
PWM
MOSI
22
GPIO23
I2C SCL
1
TX
3
RX
21
GPIO21
I2C SDA
GND
GND
19
GPIO19
PWM
MISO
18
GPIO18
PWM
CLK
5
GPIO5
PWM
17
GPIO17
PWM
TX 2
16
GPIO16
PWM
RX 2
4
GPIO4
0
GPIO0
2
GPIO2
15
GPIO15
SD1
GPIO8
SHD/SD1
SD0
GPIO7
SWD/SD0
CLK
GPIO6
CSC/CLK
ESP-WROOM-32
Reset
USB
Flash

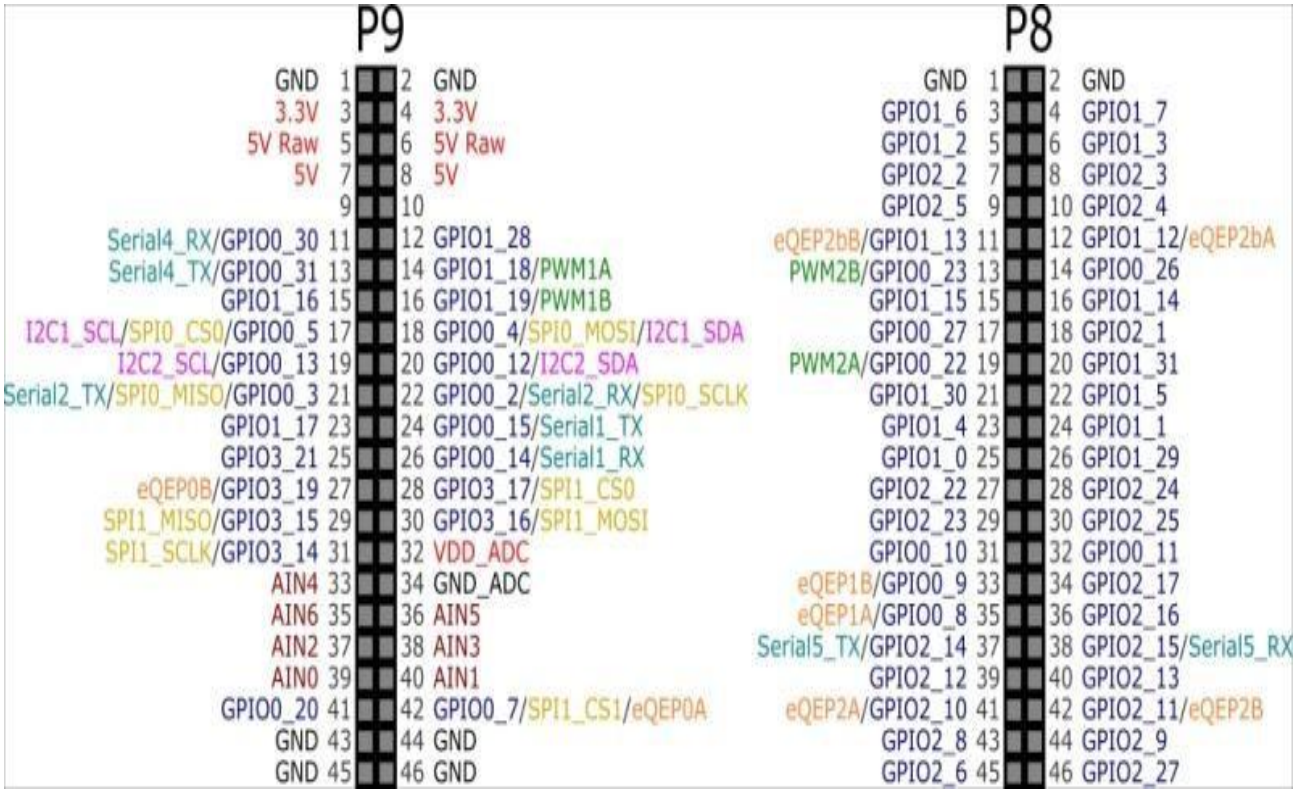
BeagleBone Black (P8/P9 Headers)

P8 Header (46-Pin):

GND —— VDD_3V3
GPIO2 —— GPIO3
GPIO4 —— GPIO5
GPIO6 —— GPIO7
GPIO8 —— GPIO9
GPIO10 —— GPIO11
GPIO12 —— GPIO13
GPIO14 —— GPIO15
GPIO16 —— GPIO17
GPIO18 —— GPIO19
GPIO20 —— GPIO21
GPIO22 —— GPIO23
GPIO24 —— GPIO25
GPIO26 —— GPIO27
GPIO28 —— GPIO29
GPIO30 —— GPIO31

P9 Header (46-Pin):

GND —— VDD_5V
SYS_5V —— VDD_3V3
VDD_ADC —— AIN0
AIN1 —— AIN2
AIN3 —— AIN4
AIN5 —— AIN6
I2C1_SCL —— I2C1_SDA
UART1_TXD —— UART1_RXD
SPI0_CS0 —— SPI0_D0
SPI0_D1 —— SPI0_SCLK



Key Pins:

- ✓ **P8_3-P8_6:** eMMC (do not use)
- ✓ **P9_19/20:** I2C2 (SCL/SDA)
- ✓ **P9_24/26:** UART1 (TXD/RXD)
- ✓ **P9_33:** 1.8V ADC (AIN4)

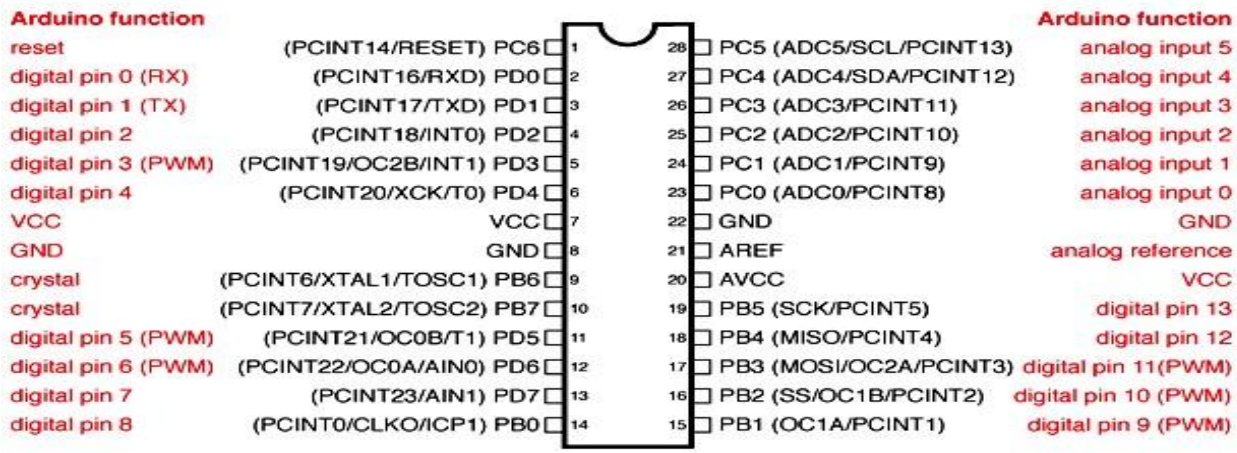
Arduino Uno R3 (ATmega328P)

Pinout:

D0 (RX) —— D1 (TX)
D2 (INT0) —— D3 (PWM)
D4 —— D5 (PWM)
D6 (PWM) —— D7
D8 —— D9 (PWM)
D10 (SS) —— D11 (MOSI/PWM)
D12 (MISO) —— D13 (SCK/LED)
A0 —— A1
A2 —— A3
A4 (SDA) —— A5 (SCL)
A6 —— A7

Key Pins:

- ✓ **A0-A5:** Analog inputs (10-bit ADC)
- ✓ **D3/D5/D6/D9/D10/D11:** PWM (8-bit)
- ✓ **D10-D13:** SPI (SS/MOSI/MISO/SCK)
- ✓ **A4/A5:** I2C (SDA/SCL)



Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

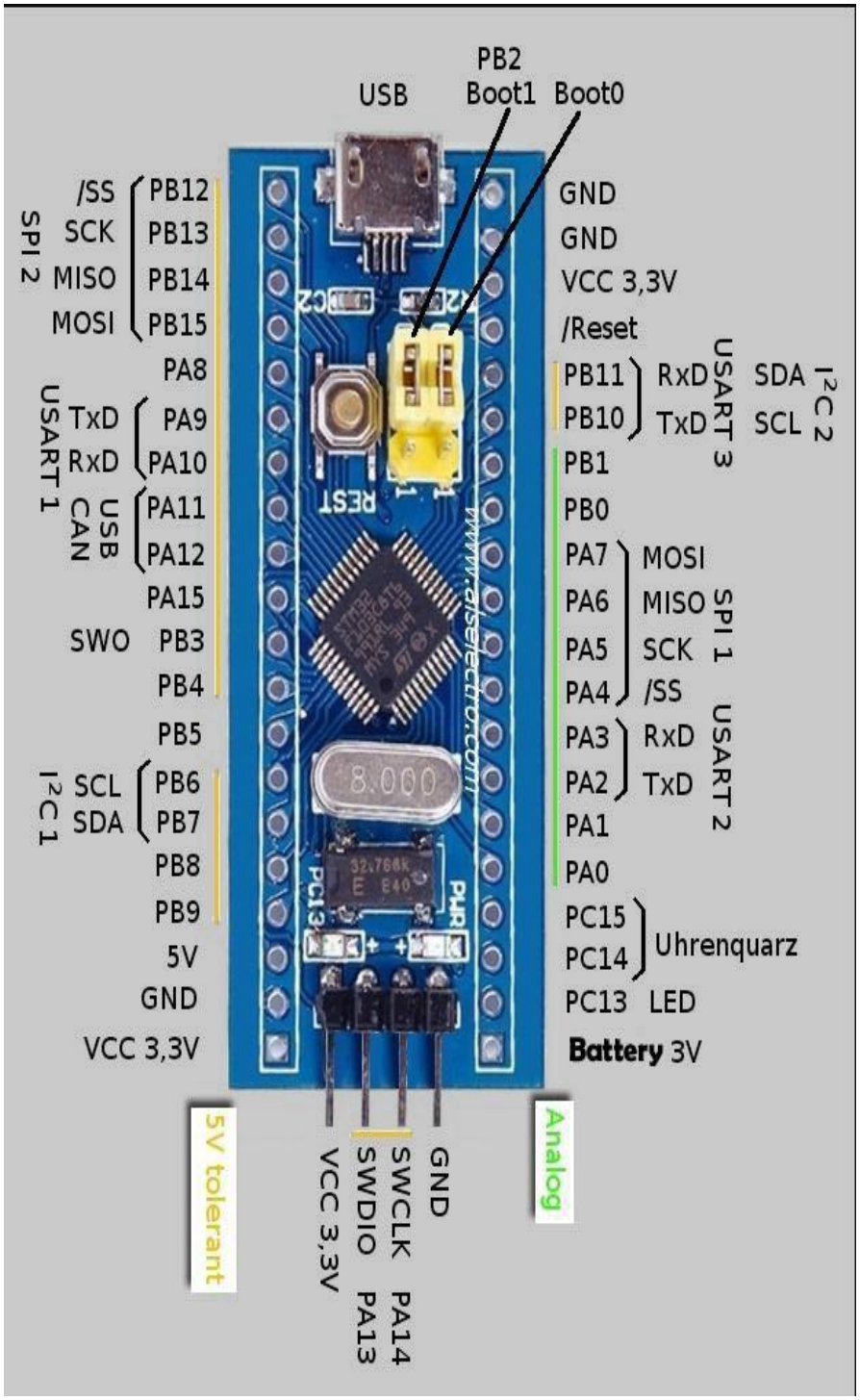
STM32F103C8T6 (Blue Pill)

Pinout:

PA0 —— PA1
PA2 —— PA3
PA4 —— PA5
PA6 —— PA7
PB0 —— PB1
PB10 —— PB11
PB12 —— PB13
PB14 —— PB15
PC13 —— PC14
GND —— 3V3

Key Pins:

- **PA9/PA10:** USART1 (TXD/RXD)
- **PB6/PB7:** I2C1 (SCL/SDA)
- **PA2/PA3:** USART2 (TXD/RXD)
- **PA0-PA7:** 12-bit ADC


USB
PB2
Boot1
Boot0
/SS
SCK
MISO
MOSI
SPI 2
PB12
PB13
PB14
PB15
PA8
TxD
RxD
USART 1
PA9
PA10
USB
CAN
PA11
PA12
PA15
SWO
PB3
PB4
PB5
SCL
SDA
I²C 1
PB6
PB7
PB8
PB9
5V
GND
VCC 3,3V
GND
GND
VCC 3,3V
/Reset
PB11
PB10
RxD
TxD
USART 3
SDA
SCL
I²C 2
PB1
PB0
PA7
PA6
PA5
PA4
MOSI
MISO
SCK
/SS
SPI 1
PA3
PA2
RxD
TxD
USART 2
PA1
PA0
PC15
PC14
Uhrenquarz
PC13
LED
Battery 3V
RESET
PWR
www.raselectro.com
5V tolerant
VCC 3,3V
SWDIO
PA13
SWCLK
PA14
GND
Analog

ARM Cortex-M (Generic Pinout)

Typical GPIO Layout:

VDD —— VSS
PA0 —— PA1
PB0 —— PB1
PC0 —— PC1
PD0 —— PD1
NRST —— BOOT0

Key Pins:

- **SWDIO/SWCLK:** Debugging (PA13/PA14)
- **USART_TX/RX:** Default serial (PA9/PA10)
- **I2C_SCL/SDA:** PB6/PB7

Raspberry Pi 4 (Broadcom BCM2711)

Pinout Diagram

https://i.imgur.com/V0pL6Rz.png (Source: Raspberry Pi Foundation)

Key Pins (40-pin GPIO):

Pin(s)	Function
1, 17	3.3 V Power
2, 4	5 V Power
3, 5	I^2C (SDA, SCL)
8, 10	UART (TX, RX)

Specifications

- **SoC**: Broadcom BCM2711 (ARM Cortex-A72)
- **CPU**: Quad-core, 64-bit @ 1.5 GHz
- **RAM**: 2 GB / 4 GB / 8 GB LPDDR4
- **GPIO**:
  - 3.3 V logic
  - 5 V tolerance **only with level shifter**
  - Multiple interfaces: I^2C, SPI, UART, PWM, GPIO

Recommended Use Cases

- Linux-based embedded systems
- Media centers (e.g., Kodi)
- Robotics and automation
- IoT gateways and protocol bridging
- Educational tools and system emulation

BeagleBone Black (TI AM3358)

Pinout Diagram

https://i.imgur.com/8JkQY7T.png *(Source: BeagleBoard.org)*

Key Header Pins (P8 & P9)

Header Pin	Function	Purpose
P8.3	GPIO1_6	eMMC disable (boot override)
P9.11	UART4_RXD	Serial communication (RX)
P9.14	EHRPWM1A	PWM output (motor/signal control)

Specifications

- **SoC**: Texas Instruments **AM3358**
- **CPU**: ARM Cortex-A8 @ 1 GHz
- **RAM**: 512 MB DDR3
- **PRUs**: 2 × 200 MHz Programmable Real-Time Units (PRU-ICSS)
- **GPIO Logic Level**: 3.3 V

Recommended Use Cases

- Real-time motor and sensor control
- Industrial automation and control panels
- Embedded Linux systems with deterministic behavior
- PRU-based I/O interfacing (SPI, I^2C, PWM, etc.)

nRF52840 (Bluetooth MCU)

Pinout Diagram

Source: Nordic Semiconductor

Pin(s)	Function
P0.01–P1.15	GPIO, ADC, NFC, I^2C, SPI
VDD	3.3 V Power

Specs:

- **MCU**: ARM Cortex-M4F @ 64 MHz
- **Wireless**: Bluetooth 5.2, Thread, Zigbee
- **Memory**: 1 MB Flash / 256 KB RAM
- **Logic Level**: 1.7 V – 3.6 V

Use Cases:

- Wearables, Bluetooth LE mesh networks, IoT sensors

PIC16F877A (8-bit MCU)

📷 Pinout Diagram
Source: Microchip

Pin(s)	Function
RA0–RA5	Analog Input (ADC)
RB0–RB7	Digital I/O
RC6–RC7	UART TX/RX

Specs:

- **MCU**: 8-bit PIC @ 20 MHz
- **Memory**: 14 KB Flash / 368 B RAM
- **Operating Voltage**: 5 V

Use Cases:

- Legacy industrial control, educational microcontroller systems

Xilinx Zynq-7000 (FPGA + SoC)

📷 *Pinout varies by board* (e.g. Zybo Z7)

Type	Feature
ARM SoC	Dual-core Cortex-A9 @ up to 1 GHz
FPGA Fabric	Artix-7 class programmable logic
I/O Banks	1.8 V – 3.3 V, High-Speed Serial, LVDS, etc.

Use Cases:

- Embedded vision, aerospace, SDR, high-speed DSP, tightly integrated control + acceleration tasks

Summary Table

Platform	Core	Clock	Memory	Main Use Cases
nRF52840	Cortex-M4F	64 MHz	1 MB Flash / 256 KB RAM	Wearables, BLE Mesh, Zigbee, low-power IoT
PIC16F877A	8-bit PIC	20 MHz	14 KB Flash / 368 B RAM	Legacy industrial, education
Zynq-7000	Cortex-A9 + FPGA	650–1000 MHz	Varies by board	High-speed DSP, FPGA-accelerated SoC apps