
CSPC61, EMBEDDED SYSTEMS AND ARCHITECTURE
CHAPTER-8: DEVICE DRIVERS

---

1. What is a device driver?

Device drivers are the software libraries that initialize the hardware and manage access to the hardware by higher layers of software. It directly interfaces with and controls the hardware. Device drivers are the liaison between the hardware and the operating system, middleware, and application layers.

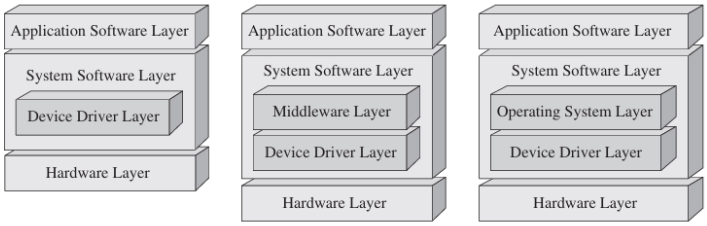

Figure 8-1
Embedded Systems Model and Device Drivers.

---

2. Which of Figures 8-46a, b, c, and d is incorrect in terms of mapping device driver software into the Embedded Systems Model? **Figure 8-46b**

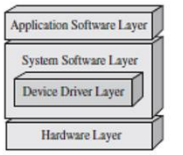

Figure 8-46a: Example 1

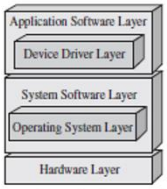

Figure 8-46b: Example 2

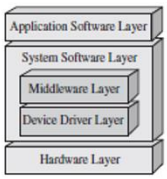

Figure 8-46c: Example 3

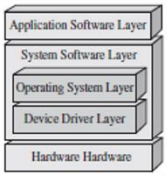

Figure 8-46d: Example 4

---

3. What is the difference between an architecture-specific device driver and a generic device driver? Give two examples of each.

Parameter	*Architecture-Specific*	*Generic*
Functionality	Manages hardware integrated into the master processor (the architecture).	Manages hardware located on the board and not integrated onto the master processor.
Code Structure	Primarily involves low-level hardware access functions optimized for the specific processor architecture.	Typically includes architecture-specific portions of source code, as the master processor serves as the central control unit to access anything on the board.
Versatility	Limited to the architecture it's designed for.	Can be configured to run on a variety of architectures that contain the related board hardware for which the driver is written.
Examples	• On-Chip memory • Integrated memory managers (MMUs) • Floating-point hardware	• Board buses (I2C, PCI, PCMCIA) • Off-chip memory (controllers, cache, Flash, etc) • Off-chip I/O (Ethernet, RS-232, display, mouse, etc)

---

5. List and describe five types of device driver functions.

i. *Hardware Startup*: Initialization of the hardware upon Power-ON or reset.
ii. *Hardware Shutdown*: Configuring hardware into its Power-OFF state.
iii. *Hardware Disable*: Allowing other software to disable hardware on-the-fly.
iv. *Hardware Enable*: Allowing other software to enable hardware on-the-fly.
v. *Hardware Acquire*: Allowing other software to gain singular (locking) access to hardware.

---

6. Finish the sentence: The software's implicit perception of hardware is that it exists in one of three states at any given time:

inactive, finished, or busy / inactive, finished, or broken / fixed, finished, or busy / fixed, inactive, or broken / None of the above.

---

7. [T/F] On master processors that offer different modes in which different types of software can execute, device drivers usually do not run in supervisory mode. **False**.

Depending on the master processor, different types of software can execute in different modes, the most common being supervisory and user modes. These modes essentially differ in terms of what system components the software is allowed access to, with software running in supervisory mode having more access (privileges) than software running in user mode. **Device driver code typically runs in supervisory mode.**

---

8. What is an interrupt? How can interrupts be initiated?

Interrupts are signals triggered by some event during the execution of an instruction stream by the master processor.

Interrupts can be initiated asynchronously, for external hardware devices, resets, power failures, etc., or synchronously, for instruction-related activities such as system calls or illegal instructions. These signals cause the master processor to stop executing the current instruction stream and start the process of handling (processing) the interrupt.

---

9. Name and describe four examples of device driver functions that can be implemented for interrupt handling.

i. *Interrupt Handling Startup*: Initialization of the interrupt hardware (interrupt controller, activating interrupts, etc.) upon Power-ON or reset.
ii. *Interrupt Handling Shutdown*: Configuring interrupt hardware (interrupt controller, deactivating interrupts, etc.) into its Power-OFF state.
iii. *Interrupt Handling Disable*: Allowing other software to disable active interrupts on-the fly (not allowed for non-maskable interrupts (NMIs), which are interrupts that cannot be disabled).
iv. *Interrupt Handling Enable*: Allowing other software to enable inactive interrupts on-the-fly.
v. *Interrupt Handler Servicing*: The interrupt handling code itself, which is executed after the interruption of the main execution stream (this can range in complexity from a simple non-nested routine to nested and/or re-entrant routines).

---

10. What are the three main types of interrupts? List examples in which each type is triggered.

The three main types of interrupts are software, internal hardware, and external hardware.

i. *Software Interrupts*: These are explicitly triggered internally by some instruction within the current instruction stream being executed by the master processor.
ii. *Internal Hardware Interrupts*: These, on the other hand, are initiated by an event that is a result of a problem with the current instruction stream that is being executed by the master processor because of the features (or limitations) of the hardware, such as illegal math

---

operations (overflow, divide-by-zero), debugging (single-stepping, breakpoints), and invalid instructions (opcodes).

iii. *External Hardware Interrupts*: These are interrupts initiated by hardware other than the master CPU (board buses, I/O, etc.).

---

11. What is the difference between a level-triggered interrupt and an edge-triggered interrupt? What are some strengths and drawbacks of each?

Parameter	*Level-Triggered Interrupt*	*Edge-Triggered Interrupt*
Initiation	Initiated when IRQ signal remains at a certain level (HIGH or LOW).	Triggered by a change in the IRQ signal (LOW to HIGH or vice versa).
Processing	Processed when CPU samples IRQ line for a level-triggered request.	Latches into the CPU until processed once triggered by signal change.
Strengths	Suitable for interrupts with shared IRQ lines.	Suitable for short or long interrupt signals.
Drawbacks	May cause CPU to repeatedly service the same interrupt.	Risk of missing interrupts if triggered simultaneously on shared IRQ lines.

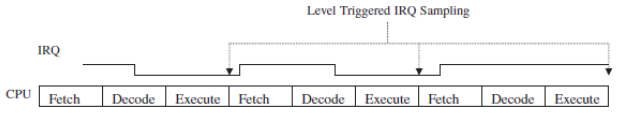

Figure 4-64a: Level-triggered interrupts [4-24]

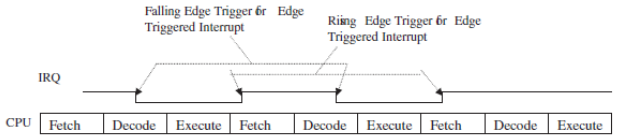

Figure 4-64b: Edge-triggered interrupts [4-24]

---

12. An IACK is: An interrupt controller / An IRQ port / **An interrupt acknowledgement** / None.

---

13. [T/F] An ISR is executed before an interrupt is triggered.

False. An ISR (Interrupt Service Routine) is executed in response to an interrupt being triggered, not before.

---

14. What is the difference between an auto-vectored and an interrupt-vectored scheme?

Parameter	*Interrupt-Vectored Scheme*	*Auto-Vectored Scheme*
Interrupt Vector	Depends on external device providing an interrupt vector.	Not applicable (Used when devices cannot provide interrupt vectors).
ISR Handling	Each device has its own interrupt vector pointing to its specific ISR.	A single ISR is shared among all non-vectored interrupts.
Responsibility	CPU directly jumps to the appropriate ISR using the interrupt vector.	ISR handles determining which specific interrupt occurred and other related tasks internally.
Implementation	Used when devices can provide interrupt vectors.	Used when devices cannot provide interrupt vectors (non-vectored interrupts).

---

15. Name and describe four examples of device driver functions that can be implemented for managing memory.

i. *Memory Subsystem Startup*: Initialization of the hardware upon Power-ON or reset (initialize translation lookaside buffers (TLBs) for MMU, initialize/configure MMU).
ii. *Memory Subsystem Shutdown*: Configuring hardware into its Power-OFF state. (Note: Under the MPC860, there is no necessary shutdown sequence for the memory subsystem, so pseudocode examples are not shown.)
iii. *Memory Subsystem Disable*: Allowing other software to disable hardware on-the-fly (disabling cache).
iv. *Memory Subsystem Enable*: Allowing other software to enable hardware on-the-fly (enable cache).
v. *Memory Subsystem Write*: Storing in memory a byte or set of bytes (i.e., in cache, ROM, and main memory).
vi. *Memory Subsystem Read*: Retrieving from memory a "copy" of the data in the form of a byte or set of bytes (i.e., in cache, ROM, and main memory).

---

16. What is byte ordering? Name and describe the possible byte ordering schemes.

The order in which bytes are retrieved or stored in memory is known as byte ordering. It depends on the byte ordering scheme of an architecture.

The two possible byte ordering schemes are little-endian and big endian. In little-endian mode, bytes (or "bits" with 1-byte (8-bit) schemes) are retrieved and stored in the order of the lowest byte first, meaning the lowest byte is furthest to the left. In big-endian mode bytes are accessed in the order of the highest byte first, meaning that the lowest byte is furthest to the right.

	Odd Bank	Even Bank	
F	90	87	E
D	E9	11	C
8	F1	24	A
9	01	46	8
7	76	DE	6
5	14	33	4
3	55	12	2
1	AB	FF	0
	⇩	⇩	
	Data Bus (15:8)	Data Bus (7:0)	

In little-endian mode if a byte is read from address "0", an "FF" is returned; if 2 bytes are read from address 0, then (reading from the lowest byte which is furthest to the LEFT in little-endian mode) an "ABFF" is returned. If 4 bytes (32-bits) are read from address 0, then a "5512ABFF" is returned.

In big-endian mode if a byte is read from address "0", an "FF" is returned; if 2 bytes are read from address 0, then (reading from the lowest byte which is furthest to the RIGHT in big-endian mode) an "FFAB" is returned. If 4 bytes (32-bits) are read from address 0, then a "1255FFAB" is returned.

---

17. Name and describe four examples of device driver functions that can be implemented for bus protocols.

i. *Bus Startup*: Initialization of the bus upon Power-ON or reset.
ii. *Bus Shutdown*: Configuring bus into its Power-OFF state.
iii. *Bus Disable*: Allowing other software to disable bus on-the-fly.
iv. *Bus Enable*: Allowing other software to enable bus on-the-fly.

---

18. Name and describe four examples of device driver functions that can be implemented for I/O.

i. *I/O Startup*: Initialization of the I/O upon Power-ON or reset.
ii. *I/O Shutdown*: Configuring I/O into its Power-OFF state.
iii. *I/O Disable*: Allowing other software to disable I/O on-the-fly.
iv. *I/O Enable*: Allowing other software to enable I/O on-the-fly.

---

19. Where in the OSI model are the Ethernet and serial device drivers mapped to?

They are mapped to the lower section of the OSI (Open Systems Interconnection) data-link layer.
