0001 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：基礎

単一コアRTOSで、実行可能だが同じまたは高い優先度の別タスクがCPUを使用しているため実行されていないタスクの状態として最も適切なものはどれか。

ア．Ready状態
イ．Blocked状態
ウ．Suspended状態
エ．Running状態

答え・解説 正答：ア

ア：Ready状態は実行可能条件を満たしているが、現在CPUを割り当てられていない状態である。
イ：Blocked状態は時間やイベントを待っており、現時点では実行可能でない。
ウ：Suspended状態は明示的な再開操作があるまでスケジューリング対象にならない。
エ：Running状態は実際にCPU上で命令を実行している状態である。
総合解説：Ready状態は実行可能条件を満たしているが、現在CPUを割り当てられていない状態である。 この問題では「RTOSタスク状態」として、Running状態・Ready状態・Blocked状態の違いを判断できる。

対象：2026年度 / 基準日 2026-09-02

0002 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：基礎

キュー受信待ちでBlocked状態になっているタスクについて正しい説明はどれか。

ア．待っている間も最高優先度でCPUを占有し続ける。
イ．待っている間はCPU時間を消費せず、データ到着またはタイムアウトなどでReady状態へ移る。
ウ．Blocked状態へ入るとタスクは必ず削除される。
エ．Blocked状態にはタイムアウトを設定できない。

答え・解説 正答：イ

ア：Blocked状態は実行対象にならない。
イ：FreeRTOSではBlocked状態のタスクはCPU時間を消費せず、待機条件成立やタイムアウトで再び実行可能になる。
ウ：待機状態であり削除とは異なる。
エ：多くの待機APIではタイムアウトを指定できる。
総合解説：FreeRTOSではBlocked状態のタスクはCPU時間を消費せず、待機条件成立やタイムアウトで再び実行可能になる。 この問題では「イベント待ちによるブロック」として、Blocked状態のタスクがCPU時間を消費しないことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0003 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：基礎

FreeRTOSのSuspended状態について最も適切な説明はどれか。

ア．一定時間経過すると必ず自動でRunningになる。

イ．Ready状態と同じ意味である。

ウ．明示的に再開されるまでスケジューラから選択されず、通常のBlocked状態のようなタイムアウトでは復帰しない。

エ．割込みが発生するたびに必ず削除される。

答え・解説 正答：ウ

ア：Suspended状態には通常の時間待ちによる自動解除はない。

イ：Readyは実行可能、Suspendedは実行不可である。

ウ：Suspended状態はvTaskSuspend()等で明示的に入り、xTaskResume()等で再開されるまで実行対象にならない。

エ：割込み発生とタスク削除は別である。

総合解説：Suspended状態はvTaskSuspend()等で明示的に入り、xTaskResume()等で再開されるまで実行対象にならない。 この問題では「BlockedとSuspended比較」として、Suspended状態とBlocked状態のタイムアウト有無の違いを理解できる。

対象：2026年度 / 基準日 2026-09-02

0004 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：標準

タスクAがセマフォ取得待ちでBlocked状態にある。ISRがセマフォを与え、Aが実行可能になった。この直後のAの基本的な状態遷移はどれか。

ア．Blockedから必ずSuspendedへ移る。

イ．BlockedからDeletedへ移る。

ウ．Blockedのまま永久に残る。

エ．BlockedからReadyへ移り、優先度条件を満たせばRunningへ遷移する。

答え・解説 正答：エ

ア：イベント成立でSuspendedになるわけではない。

イ：セマフォ成立は削除条件ではない。

ウ：待機条件が成立しているため実行可能になる。

エ：待機イベント成立でまず実行可能状態になり、その後スケジューリングによりCPUを得る。

総合解説：待機イベント成立でまず実行可能状態になり、その後スケジューリングによりCPUを得る。 この問題では「状態遷移」として、イベント待ちタスクの状態遷移を説明できる。

対象：2026年度 / 基準日 2026-09-02

0005 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：標準

RTOSがタスクAからタスクBへコンテキストスイッチするとき、Aを後で同じ位置から再開するために必要な処理はどれか。

ア．AのCPUレジスタなどの実行コンテキストを保存し、Bの保存済みコンテキストを復元する。
イ．Aのソースコードを削除する。
ウ．全RAMを初期化する。
エ．Bの優先度を必ず0へ変更する。

答え・解説 正答：ア

ア：タスク切替では各タスクのレジスタ値やスタック等を保存・復元して実行を継続する。
イ：実行切替とソースコード削除は無関係である。
ウ：タスク状態を破壊してしまう。
エ：コンテキストスイッチ自体にそのような要件はない。
総合解説：タスク切替では各タスクのレジスタ値やスタック等を保存・復元して実行を継続する。 この問題では「コンテキストスイッチ」として、コンテキストスイッチ時にタスク固有の実行状態が保存・復元されることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0006 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：標準

100ms周期で処理を繰り返すタスクの長期的な周期ずれを抑えたい。FreeRTOSで適した考え方はどれか。

ア．各処理終了後に常に100msの相対遅延だけを入れる。
イ．前回の基準時刻から次回の絶対起床時刻を計算するxTaskDelayUntil()系を用いる。
ウ．待機せず無限ループで時刻を監視する。
エ．タスクを毎周期削除して再作成する。

答え・解説 正答：イ

ア：処理時間分が周期へ加算され、長期的にずれやすい。
イ：絶対時刻基準で次回起床を決めるため、処理時間のばらつきが周期へ累積しにくい。
ウ：CPUを浪費し、他タスクの実行にも悪影響を与える。
エ：不要なオーバヘッドと資源管理負荷を生む。
総合解説：絶対時刻基準で次回起床を決めるため、処理時間のばらつきが周期へ累積しにくい。 この問題では「周期タスクの遅延」として、周期タスクで相対遅延より絶対時刻基準の待機が周期ずれを抑えやすいことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0007 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：標準

全アプリケーションタスクがBlockedまたはSuspended状態になったとき、RTOSで一般に実行されるものはどれか。

ア．最も最近Blockedになったタスク
イ．必ず最高優先度タスク
ウ．Idleタスク
エ．削除済みタスク

答え・解説 正答：ウ

ア：Blocked状態のタスクは実行対象ではない。
イ：最高優先度でもBlockedなら実行できない。
ウ：実行可能なアプリケーションタスクがなければ、最低優先度のIdleタスクがCPUを利用する。
エ：削除済みタスクはスケジューリング対象ではない。
総合解説：実行可能なアプリケーションタスクがなければ、最低優先度のIdleタスクがCPUを利用する。 この問題では「Idleタスク」として、アイドルタスクが他に実行可能タスクがない場合に実行されることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0008 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：標準

通信応答を待つタスクが、相手装置故障時にも永久に停止しないようにしたい。最も適切な設計はどれか。

ア．無限待ちだけを使い、相手装置は必ず応答すると仮定する。
イ．待機中も最高優先度でビジーループする。
ウ．応答確認を行わず成功扱いする。
エ．待機APIに有限のタイムアウトを設定し、期限超過時に異常処理や再試行へ遷移する。

答え・解説 正答：エ

ア：相手故障時にタスクが永久待機する。
イ：CPUを浪費し他処理を妨げる。
ウ：通信異常を検出できない。
エ：有限タイムアウトを設けるとイベントが来ない故障時にも制御を回収できる。
総合解説：有限タイムアウトを設けるとイベントが来ない故障時にも制御を回収できる。 この問題では「待機タイムアウト」として、タスク待機でタイムアウトを設ける意義を判断できる。

対象：2026年度 / 基準日 2026-09-02

0009 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：応用

小さな処理ごとに100個のタスクを作成したところ、RAM不足とコンテキストスイッチ増加が問題になった。改善方針として最も適切なものはどれか。

ア．独立した周期・優先度・待機条件が必要な処理を基準にタスク責務を再整理し、不要に細かいタスクを統合する。

イ．さらにタスクを細分化して1000個にする。

ウ．全タスクを最高優先度にする。

エ．スタック不足を無視して運用する。

答え・解説 正答：ア

ア：各タスクは独自スタックなどの資源を持つため、過度な細分化はRAMと切替オーバヘッドを増やす。

イ：資源問題を悪化させる可能性が高い。

ウ：優先度設計が崩れ、問題解決にならない。

エ：メモリ破壊や不安定動作の原因になる。

総合解説：各タスクは独自スタックなどの資源を持つため、過度な細分化はRAMと切替オーバヘッドを増やす。 この問題では「タスク分割粒度」として、タスク分割の粒度がスタック・切替オーバヘッドと責務分離に影響することを評価できる。

対象：2026年度 / 基準日 2026-09-02

0010 3-1 RTOS・タスク管理 > タスク・状態遷移 ｜ 難易度：応用

センサデータは1秒に数回しか到着しない。受信タスクがwhileループで常時フラグを監視してCPU使用率が高い。最も適切な改善はどれか。

ア．監視ループをさらに高速化する。

イ．キュー・通知・セマフォなどのイベント待ちでBlocked状態にし、データ到着時だけReadyにする。

ウ．受信タスクを最高優先度のまま無限ループさせる。

エ．センサデータを受信しない。

答え・解説 正答：イ

ア：CPU使用率をさらに上げる可能性がある。

イ：イベント待ちにより不要なポーリングを避け、CPU時間と消費電力を他処理へ回せる。

ウ：他タスクを飢餓状態にし得る。

エ：機能要件を満たさない。

総合解説：イベント待ちにより不要なポーリングを避け、CPU時間と消費電力を他処理へ回せる。 この問題では「イベント駆動設計」として、イベント駆動タスクでポーリングよりブロック待ちを用いる利点を評価できる。

対象：2026年度 / 基準日 2026-09-02

0011 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：基礎

固定優先度プリエンプティブRTOSで、現在実行中より高優先度のタスクがReadyになった場合の一般的な動作はどれか。

ア．現在のタスクが永久に実行を続ける。
イ．両タスクが単一コア上で同時に命令を実行する。
ウ．高優先度タスクが現在のタスクをプリエンプトして実行される。
エ．高優先度タスクが自動削除される。

答え・解説 正答：ウ

ア：高優先度タスクの即時実行性が失われる。
イ：単一コアでは同時実行できない。
ウ：プリエンプティブスケジューリングでは、より高優先度のReadyタスクが実行権を得る。
エ：Ready化は削除条件ではない。
総合解説：プリエンプティブスケジューリングでは、より高優先度のReadyタスクが実行権を得る。 この問題では「最高優先度Readyタスク」として、固定優先度プリエンプティブスケジューリングの基本を理解できる。

対象：2026年度 / 基準日 2026-09-02

0012 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：基礎

タイムスライシングが有効なRTOSで、同一の最高優先度を持つ複数タスクがReady状態にある場合の動作として適切なものはどれか。

ア．最初のタスクだけが永久に実行されることが仕様で保証される。
イ．全タスクが自動的に優先度0になる。
ウ．Blocked状態のタスクだけが交替する。
エ．時間片ごとに実行権を交替させ、CPU時間を共有できる。

答え・解説 正答：エ

ア：タイムスライシング有効時の説明ではない。
イ：優先度は変更されない。
ウ：Blockedタスクは実行対象でない。
エ：FreeRTOSでは設定により同一優先度Readyタスク間でtickごとにタイムスライスを行える。
総合解説：FreeRTOSでは設定により同一優先度Readyタスク間でtickごとにタイムスライスを行える。 この問題では「タイムスライス」として、同一優先度タスクのタイムスライスを理解できる。

対象：2026年度 / 基準日 2026-09-02

0013 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：基礎

独立な周期タスクにRate Monotonic方式で固定優先度を割り当てるとき、基本的な優先度規則はどれか。

ア．周期が短いタスクほど高い優先度を与える。
イ．周期が長いタスクほど必ず高優先度にする。
ウ．全タスクを同一優先度にする。
エ．実行時間だけで優先度を決め、周期を無視する。

答え・解説 正答：ア

ア：Rate Monotonicでは要求頻度が高い、すなわち周期が短いタスクほど高優先度とする。
イ：Rate Monotonicとは逆である。
ウ：固定優先度割当の特徴を失う。
エ：Rate Monotonicは周期を基準にする。
総合解説：Rate Monotonicでは要求頻度が高い、すなわち周期が短いタスクほど高優先度とする。 この問題では「Rate Monotonic」として、Rate Monotonicで周期の短いタスクほど高優先度となることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0014 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：標準

最高優先度タスクが一度もBlockedせず無限ループで計算を続ける。固定優先度プリエンプティブRTOSで起こり得る問題はどれか。

ア．低優先度タスクの優先度が必ず自動上昇する。
イ．低優先度タスクがCPUを得られずStarvationに陥る。
ウ．全タスクが同時実行される。
エ．最高優先度タスクが自動的にSuspendedになる。

答え・解説 正答：イ

ア：一般のスケジューリングで自動救済されるとは限らない。
イ：最高優先度タスクが常にReadyなら、低優先度タスクの実行機会がなくなる可能性がある。
ウ：単一コアでは同時実行できない。
エ：そのような自動動作は一般にはない。
総合解説：最高優先度タスクが常にReadyなら、低優先度タスクの実行機会がなくなる可能性がある。 この問題では「Starvation」として、優先度の高いCPU占有タスクが低優先度タスクを飢餓状態にすることを判断できる。

対象：2026年度 / 基準日 2026-09-02

0015 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：標準

周期10ms・実行時間2msのタスクAと、周期20ms・実行時間5msのタスクBがある。単純なCPU利用率の合計はいくらか。

ア．25%（タスクBだけを計上）
イ．70%（実行時間を別比率で換算）
ウ．45%（2/10＋5/20）
エ．140%（周期比を逆向きに換算）

答え・解説 正答：ウ

ア：タスクAの利用率を加えていない。
イ：計算結果と一致しない。
ウ：Aは2/10=20%、Bは5/20=25%なので合計45%である。
エ：周期に対する実行時間の比を誤っている。
総合解説：Aは2/10=20%、Bは5/20=25%なので合計45%である。 この問題では「CPU利用率計算」として、周期タスクのCPU利用率を実行時間と周期から計算できる。

対象：2026年度 / 基準日 2026-09-02

0016 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：標準

ログ保存タスクは業務上重要だが期限は1秒、モータ制御タスクは5ms以内に完了する必要がある。優先度設計として最も適切な考え方はどれか。

ア．ログが重要なので必ずログを最高優先度にする。
イ．全タスクを同じ優先度にすれば期限は必ず守られる。
ウ．優先度はタスク名の文字数で決める。
エ．業務上の重要度だけでなく、デッドライン・周期・ブロッキング時間を基にモータ制御へ十分高い優先度を与える。

答え・解説 正答：エ

ア：時間制約を無視すると制御期限を違反する可能性がある。
イ：時間制約の差を反映できない。
ウ：リアルタイム要件と無関係である。
エ：リアルタイム優先度は時間制約を満たすために設計する必要がある。
総合解説：リアルタイム優先度は時間制約を満たすために設計する必要がある。 この問題では「優先度設計」として、固定優先度設計で重要度ではなく期限・周期など時間要件を考慮できる。

対象：2026年度 / 基準日 2026-09-02

0017 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：標準

タスク内の処理時間が毎回2〜8msで変動するが、50msごとに処理開始したい。適した方法はどれか。

ア．前回の基準起床時刻に50msを加えて次回起床を決める。

イ．処理終了後に常に50ms待つ。

ウ．待機せず最高優先度でループする。

エ．50msごとにタスクを削除する。

答え・解説 正答：ア

ア：絶対時刻基準なら処理時間変動による周期ドリフトを抑えられる。

イ：実周期が52〜58ms程度となりやすい。

ウ：CPUを浪費する。

エ：不要な資源操作である。

総合解説：絶対時刻基準なら処理時間変動による周期ドリフトを抑えられる。 この問題では「周期起床」として、周期タスクで相対遅延よりxTaskDelayUntil系が一定周期に適することを判断できる。

対象：2026年度 / 基準日 2026-09-02

0018 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：標準

Liu & LaylandのRate Monotonic利用率上限による判定について最も適切な説明はどれか。

ア．上限を1%でも超えれば必ず全タスクが期限違反する。

イ．十分条件なので、上限を超えても直ちにスケジュール不可能とは限らず、より詳細な解析が必要になる。

ウ．利用率が0%でも必ず期限違反する。

エ．この上限は周期と実行時間に無関係である。

答え・解説 正答：イ

ア：十分条件を必要条件と誤解している。

イ：利用率上限以下なら一定条件下でスケジュール可能と保証できるが、上限超過が即不可を意味するわけではない。

ウ：明らかに不適切である。

エ：タスク利用率の和に基づく。

総合解説：利用率上限以下なら一定条件下でスケジュール可能と保証できるが、上限超過が即不可を意味するわけではない。 この問題では「RM利用率上限」として、Rate Monotonicの十分条件が必要条件ではないことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0019 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：応用

単一プロセッサで実行時に優先度を動的に変えるEDF方式の基本規則はどれか。

ア．周期が最長のタスクだけを常に最高優先度にする。
イ．タスク生成順だけで優先度を固定する。
ウ．絶対デッドラインが最も近い実行可能ジョブを優先する。
エ．実行時間が最長のジョブを必ず優先する。

答え・解説 正答：ウ

ア：EDFの規則ではない。
イ：デッドラインを用いた動的割当ではない。
ウ：EDFはEarliest Deadline Firstの名称どおり、最も早い期限を持つジョブを優先する動的方式である。
エ：EDFの基準は絶対デッドラインである。
総合解説：EDFはEarliest Deadline Firstの名称どおり、最も早い期限を持つジョブを優先する動的方式である。 この問題では「Earliest Deadline First」として、EDFが近い絶対デッドラインを持つジョブへ高い動的優先度を与えることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0020 3-1 RTOS・タスク管理 > スケジューリング・優先度 ｜ 難易度：応用

5ms周期の高優先度制御タスクが、平均実行時間は短いのに時々期限超過する。最も適切な調査はどれか。

ア．平均実行時間だけが短いので問題なしとする。
イ．優先度を全て同一にする。
ウ．測定せずCPUクロックだけ最大にする。
エ．より高優先度タスクやISRの干渉、共有資源のブロッキング、WCET、コンテキストスイッチ時間を含めて応答時間を分析する。

答え・解説 正答：エ

ア：最悪時の期限超過を説明できない。
イ：時間制約に基づく設計を失う。
ウ：原因を特定せず余裕だけで解決するのは不十分である。
エ：平均実行時間だけでなく、プリエンプションとブロッキングを含む最悪応答時間を評価する必要がある。
総合解説：平均実行時間だけでなく、プリエンプションとブロッキングを含む最悪応答時間を評価する必要がある。 この問題では「優先度設計の総合評価」として、優先度設計でISR・共有資源・実行時間を含めて最悪応答を評価できる。

対象：2026年度 / 基準日 2026-09-02

0021 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：基礎

複数タスクが共有I2Cドライバを同時操作しないよう排他制御したい。FreeRTOSで一般に適した同期オブジェクトはどれか。

ア．Mutex
イ．Binary Semaphoreだけを必ず使う
ウ．Idleタスク
エ．ウォッチドッグ

答え・解説 正答：ア

ア：Mutexは共有資源の相互排他を目的とし、優先度継承機構も持つ。
イ：同期通知には適するが、単純な相互排他ではMutexの優先度継承が有利である。
ウ：排他制御オブジェクトではない。
エ：ソフトウェア監視用で排他機構ではない。
総合解説：Mutexは共有資源の相互排他を目的とし、優先度継承機構も持つ。 この問題では「MutexとSemaphore」として、MutexとBinary Semaphoreの代表的用途を区別できる。

対象：2026年度 / 基準日 2026-09-02

0022 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：基礎

低優先度タスクがMutexを保持中に高優先度タスクが同じMutexを待った。優先度継承の基本動作はどれか。

ア．高優先度タスクの優先度を永久に最低へ下げる。
イ．Mutex保持中の低優先度タスクの実効優先度を一時的に引き上げ、資源解放を早める。
ウ．Mutexを自動削除する。
エ．全タスクをSuspendedにする。

答え・解説 正答：イ

ア：優先度継承とは逆である。
イ：優先度継承により中優先度タスクの割込みで低優先度保持者が長く遅延する影響を抑える。
ウ：資源保護機能を失う。
エ：Priority Inversion解消方法ではない。
総合解説：優先度継承により中優先度タスクの割込みで低優先度保持者が長く遅延する影響を抑える。 この問題では「Priority Inheritance」として、Mutexの優先度継承がPriority Inversionの影響を軽減することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0023 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：基礎

センサISRから受信タスクへ小さなイベントデータを順序付きで渡したい。適した仕組みはどれか。

ア．同じグローバル変数を無保護で上書きする。
イ．MutexをISRで取得して長時間待つ。
ウ．RTOS Queue
エ．全イベントを無条件に破棄する。

答え・解説 正答：ウ

ア：イベントの取りこぼしや競合が起こり得る。
イ：ISRはMutex待ちでブロックすべきでない。
ウ：Queueはタスク間およびISRからタスクへのメッセージ受渡しに利用でき、FIFOとして順序を保持できる。
エ：機能要件を満たさない。
総合解説：Queueはタスク間およびISRからタスクへのメッセージ受渡しに利用でき、FIFOとして順序を保持できる。 この問題では「RTOS Queue」として、Queueがタスク間・ISR-タスク間通信でデータをFIFO管理できることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0024 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：標準

同時に3個まで使用できる同一種類のバッファ資源を複数タスクへ貸し出したい。適した仕組みはどれか。

ア．Binary Semaphoreだけで資源数3を直接表す。
イ．Idleタスクの優先度を3にする。
ウ．スタックサイズを3バイトにする。
エ．最大カウント3のCounting Semaphore

答え・解説 正答：エ

ア：二値だけでは3個の資源数を直接管理できない。
イ：資源個数管理とは無関係である。
ウ：資源管理機能ではない。
エ：利用可能資源数をカウントで表し、取得・返却に応じて増減させることができる。
総合解説：利用可能資源数をカウントで表し、取得・返却に応じて増減させることができる。 この問題では「Counting Semaphore」として、Counting Semaphoreを同種資源の個数管理へ利用できる。

対象：2026年度 / 基準日 2026-09-02

0025 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：標準

タスクAがMutex Xを保持してYを待ち、タスクBがMutex Yを保持してXを待っている。発生している状態はどれか。

ア．Deadlock
イ．Starvationだけ
ウ．Priority Inheritance
エ．Context Switch

答え・解説 正答：ア

ア：互いに相手が保持する資源の解放を待ち、どちらも進行できない循環待ちである。
イ：Starvationは実行機会を長期間得られない状態で、ここでは資源の循環待ちが直接原因である。
ウ：優先度継承そのものではない。
エ：単なるタスク切替とは異なる。
総合解説：互いに相手が保持する資源の解放を待ち、どちらも進行できない循環待ちである。 この問題では「Deadlock」として、Deadlockの成立例を識別できる。

対象：2026年度 / 基準日 2026-09-02

0026 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：標準

複数タスクがMutex AとBの両方を必要とする。Deadlock予防策として有効なものはどれか。

ア．タスクごとにランダムな順序で取得する。
イ．全タスクでMutexを取得する順序をA→Bのように統一する。
ウ．Mutex解放処理を削除する。
エ．待機タイムアウトを無限大にするだけ。

答え・解説 正答：イ

ア：循環待ちを起こしやすい。
イ：資源取得順序を固定すると循環待ち条件を崩し、Deadlockを防ぎやすい。
ウ：他タスクが永久に取得できなくなる。
エ：Deadlockの構造を解消しない。
総合解説：資源取得順序を固定すると循環待ち条件を崩し、Deadlockを防ぎやすい。 この問題では「Deadlock予防」として、Mutex取得順序を統一して循環待ちを防止できる。

対象：2026年度 / 基準日 2026-09-02

0027 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：標準

割込み禁止を伴うCritical Sectionの設計として最も適切なものはどれか。

ア．ログ出力や通信完了待ちも含めて数秒間割込み禁止にする。
イ．全アプリケーションを常時Critical Sectionにする。
ウ．共有データを更新する必要最小限の処理だけを囲み、長時間I/Oや待機処理を入れない。
エ．共有資源更新中も一切保護しない。

答え・解説 正答：ウ

ア：リアルタイム応答を大きく悪化させる。
イ：プリエンプションや割込み応答を失う。
ウ：Critical Sectionが長いと割込みレイテンシや他タスク応答時間を悪化させる。
エ：競合状態を起こす可能性がある。
総合解説：Critical Sectionが長いと割込みレイテンシや他タスク応答時間を悪化させる。 この問題では「Critical Section」として、Critical Sectionを必要最小限にする理由を理解できる。

対象：2026年度 / 基準日 2026-09-02

0028 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：標準

1個のISRから1個の処理タスクへ「データ受信完了」を通知するだけで、複数メッセージを保持する必要はない。FreeRTOSで軽量な選択肢はどれか。

ア．必ず大容量Queueを1000要素作る。
イ．MutexをISRから取得して通知する。
ウ．通知せずタスクでビジーポーリングする。
エ．Direct-to-task notification

答え・解説 正答：エ

ア：要件に対して過剰でメモリを消費する。
イ：MutexはISRで使用すべきでない。
ウ：CPUを浪費する。
エ：Task notificationは特定タスクへ直接イベントを通知でき、軽量なBinary/Counting Semaphore相当として利用できる。
総合解説：Task notificationは特定タスクへ直接イベントを通知でき、軽量なBinary/Counting Semaphore相当として利用できる。 この問題では「Task Notification」として、Direct-to-task notificationが1対1通知で軽量に使えることを判断できる。

対象：2026年度 / 基準日 2026-09-02

0029 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：応用

同一タスク内で関数AがMutexを取得し、その内部から同じMutexを必要とする関数Bを呼び出す設計が避けられない。適切な方策はどれか。

ア．Recursive Mutexを用い、取得に成功した回数と同じ回数だけ解放する。

イ．通常Mutexを無条件に再取得し続ける。

ウ．MutexをISRから解放する。

エ．共有資源保護を全て削除する。

答え・解説 正答：ア

ア：再帰Mutexは所有タスクが同じMutexを複数回取得でき、最終的に取得回数分を解放して初めて他タスクが利用できる。

イ：自己Deadlockを起こす可能性がある。

ウ：Mutexの所有権と優先度継承の前提に反する。

エ：競合状態を招く。

総合解説：再帰Mutexは所有タスクが同じMutexを複数回取得でき、最終的に取得回数分を解放して初めて他タスクが利用できる。 この問題では「Recursive Mutex」として、Recursive Mutexの用途と取得回数分の解放が必要なことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0030 3-1 RTOS・タスク管理 > 同期・排他・共有資源 ｜ 難易度：応用

ISRでデータ到着を検出し、タスクへ処理開始を知らせたい。共有デバイス用Mutexは別に存在する。最も適切な設計はどれか。

ア．ISR内でMutexが空くまで無限待ちする。

イ．ISRではFromISR対応のQueue・Semaphore・Task Notification等でタスクを起床し、Mutex取得はタスク側で必要に応じて行う。

ウ．ISRで低優先度タスクを直接長時間実行する。

エ．イベントを破棄し、タスクへ通知しない。

答え・解説 正答：イ

ア：ISRでブロック待ちはできない。

イ：ISRはブロックできずMutexの優先度継承も適さないため、ISR-safeな通知APIでタスクへ処理を委譲する。

ウ：ISRはタスクではなく、長時間処理は応答性を悪化させる。

エ：処理開始要求を満たさない。

総合解説：ISRはブロックできずMutexの優先度継承も適さないため、ISR-safeな通知APIでタスクへ処理を委譲する。 この問題では「ISR同期方式」として、ISRとタスク間の同期にMutexではなくISR-safeな通知手段を選べる。

対象：2026年度 / 基準日 2026-09-02

0031 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：基礎

高頻度割込みのISR設計として最も適切なものはどれか。

ア．ISR内でファイル保存や長時間通信待ちを完了する。

イ．割込み要因をクリアせず処理を終了する。

ウ．割込み要因を確認・クリアし必要最小限のデータを保存して、重い処理はタスクへ通知して委譲する。

エ．ISRで無限ループする。

答え・解説 正答：ウ

ア：割込み占有時間が長くなり応答性を悪化させる。
イ：同じ割込みが繰り返し発生する可能性がある。
ウ：ISRを短く保つと他割込みの遅延とシステムジッタを抑えやすい。
エ：他処理が実行できなくなる。
総合解説：ISRを短く保つと他割込みの遅延とシステムジッタを抑えやすい。 この問題では「ISR短時間化」として、ISRでは長時間処理を避け、必要処理をタスクへ委譲する理由を説明できる。

対象：2026年度 / 基準日 2026-09-02

0032 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：基礎

ネスト可能な割込みコントローラで、低優先度ISRの実行中に高優先度割込みが発生した場合の一般的な動作はどれか。

ア．低優先度ISRが終了するまで高優先度割込みは必ず失われる。

イ．全割込み優先度が自動的に同一になる。

ウ．高優先度割込みが発生するとRTOSが削除される。

エ．高優先度割込みが低優先度ISRをプリエンプトして先に処理される。

答え・解説 正答：エ

ア：通常はPendingとして保持されるかプリエンプトされる。
イ：優先度設定は保持される。
ウ：そのような動作はない。
エ：NVICなどでは割込み優先度に従ってネスト処理が可能である。
総合解説：NVICなどでは割込み優先度に従ってネスト処理が可能である。 この問題では「割込みネスト」として、割込み優先度によって高優先度ISRが低優先度ISRをプリエンプトできることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0033 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：基礎

FreeRTOSのISRからQueueへデータを送信するとき、適切なAPIの考え方はどれか。

ア．xQueueSendFromISR()などISR向けAPIを使用する。
イ．タスク用のブロック時間付きAPIをISR内で無条件に使う。
ウ．Queueを使う場合は必ずスケジューラを停止する。
エ．ISRからQueueを利用することは一切できない。

答え・解説 正答：ア

ア：ISRではブロック不可など制約があるため、FromISR版APIを使用する。
イ：ISRは待機でブロックできない。
ウ：通常その必要はない。
エ：ISR-safe APIが提供されている。
総合解説：ISRではブロック不可など制約があるため、FromISR版APIを使用する。 この問題では「FromISR API」として、ISRからRTOS APIを使う場合にFromISR版を使う必要があることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0034 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：標準

ISRからQueueへデータを送り、その結果、現在実行中タスクより高優先度の受信タスクがReadyになった。適切な動作はどれか。

ア．高優先度タスクを永久に待たせる。
イ．ISR終了時に必要に応じてコンテキストスイッチし、高優先度タスクを実行する。
ウ．ISR内で受信タスクの全処理を直接実行する。
エ．Queueデータを必ず破棄する。

答え・解説 正答：イ

ア：優先度付きプリエンプションの目的に反する。
イ：FromISR APIのwokenフラグ等を利用し、割込み復帰時にスケジューラへ切替要求を伝える。
ウ：ISRとタスクの責務分離を崩す。
エ：通知目的を満たさない。
総合解説：FromISR APIのwokenフラグ等を利用し、割込み復帰時にスケジューラへ切替要求を伝える。 この問題では「ISR後のタスク切替」として、ISRで高優先度タスクを起床した場合に割込み復帰時のコンテキストスイッチが必要になることを判断できる。

対象：2026年度 / 基準日 2026-09-02

0035 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：標準

高優先度割込みの最大応答時間を短くしたい。最も重要な設計上の注意はどれか。

ア．割込み禁止区間を長くするほど応答時間は短くなる。
イ．全処理をISRへ移せば必ず改善する。
ウ．割込み禁止またはマスク区間を必要最小限にし、長いCritical Sectionを避ける。
エ．優先度設定を削除する。

答え・解説 正答：ウ

ア：関係が逆である。
イ：ISR相互の干渉を増やす可能性がある。
ウ：割込みがマスクされている間は対象ISRを開始できず、その時間が割込みレイテンシへ加算される。
エ：応答制御ができなくなる。
総合解説：割込みがマスクされている間は対象ISRを開始できず、その時間が割込みレイテンシへ加算される。 この問題では「割込みマスク時間」として、Critical Sectionの長さが割込みレイテンシへ影響することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0036 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：標準

割込みレイテンシの定義として最も適切なものはどれか。

ア．ISR開始から終了までの時間だけ。
イ．タスク作成から削除までの時間。
ウ．CPU電源投入からOS起動までの時間。
エ．割込み要求が発生してから対応ISRの実行が開始されるまでの時間。

答え・解説 正答：エ

ア：これはISR実行時間に近い。
イ：割込み応答とは無関係である。
ウ：ブート時間の説明である。
エ：割込みレイテンシには割込み禁止時間、現在命令完了、例外入口処理、高優先度ISR干渉などが影響する。
総合解説：割込みレイテンシには割込み禁止時間、現在命令完了、例外入口処理、高優先度ISR干渉などが影響する。 この問題では「割込みレイテンシ」として、割込みレイテンシをイベント発生からISR開始までの時間として理解できる。

対象：2026年度 / 基準日 2026-09-02

0037 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：標準

1kHz制御割込みと低速UART受信割込みがある。UART ISRが長く制御ISRを遅らせている。改善として適切なものはどれか。

ア．制御ISRの優先度を適切に高くし、UART ISRを短くして受信後処理をタスクへ委譲する。
イ．UART ISRをさらに長くし全ログ処理を入れる。
ウ．制御ISRを無効化する。
エ．両ISR内で無限待ちする。

答え・解説 正答：ア

ア：緊急度に応じた割込み優先度とISR短時間化を組み合わせるのが有効である。
イ：制御ISR遅延を悪化させる。
ウ：機能要件を満たさない。
エ：割込み応答が停止する。
総合解説：緊急度に応じた割込み優先度とISR短時間化を組み合わせるのが有効である。 この問題では「割込み優先度設計」として、割込み処理とタスク処理の優先度関係を総合的に設計できる。

対象：2026年度 / 基準日 2026-09-02

0038 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：標準

共有データ更新のため短時間だけスケジューリングを止めたい。設計として適切なものはどれか。

ア．数秒間の通信完了待ちを入れる。
イ．保護対象の更新だけを短く囲み、ブロックするAPIや時間の掛かるI/Oを入れない。
ウ．システム起動後ずっとスケジューラを停止する。
エ．共有データを無保護で更新する。

答え・解説 正答：イ

ア：リアルタイム応答を大きく損なう。
イ：プリエンプション停止中に長時間処理を行うと高優先度タスクの応答時間を悪化させる。
ウ：マルチタスク処理が成立しない。
エ：競合の可能性がある。
総合解説：プリエンプション停止中に長時間処理を行うと高優先度タスクの応答時間を悪化させる。 この問題では「プリエンプション制御」として、プリエンプション無効区間の必要性と危険性を評価できる。

対象：2026年度 / 基準日 2026-09-02

0039 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：応用

10kHzで発生するISRの平均実行時間が20μsである。他のオーバヘッドを無視したときISRだけのCPU利用率は何%か。

ア．2%（積を10分の1側へ換算）

イ．50%（別の負荷率を採用）

ウ．20%（10,000回×20μs/秒）

エ．200%（発生率と実行時間の積を10倍化）

答え・解説 正答：ウ

ア：10倍小さく見積もっている。

イ：計算結果と一致しない。

ウ：1秒に10,000回×20μs=200,000μs=0.2秒なので20%である。

エ：1回当たり時間と発生回数の積を誤っている。

総合解説：1秒に10,000回×20μs=200,000μs=0.2秒なので20%である。 この問題では「ISR負荷計算」として、高頻度ISRの処理量からCPU負荷を概算できる。

対象：2026年度 / 基準日 2026-09-02

0040 3-2 割込み・ドライバ > 割込み・プリエンプション ｜ 難易度：応用

CPU利用率の平均は50%程度なのに、低優先度通信タスクが時々長時間実行できない。高優先度ISRがバースト的に連続発生している。最も適切な分析はどれか。

ア．平均50%なら最悪遅延は存在しないと判断する。

イ．低優先度タスクのコードを削除する。

ウ．ISR回数の測定をやめる。

エ．平均利用率だけでなく、ISRの最大バースト数・最大実行時間・マスク時間を含めた最悪干渉時間を評価する。

答え・解説 正答：エ

ア：バースト的干渉を説明できない。

イ：機能を失う。

ウ：原因分析ができなくなる。

エ：リアルタイム応答は平均負荷ではなく最悪時の割込み集中にも影響される。

総合解説：リアルタイム応答は平均負荷ではなく最悪時の割込み集中にも影響される。 この問題では「割込み干渉分析」として、高優先度割込みの連続発生が低優先度タスクの応答へ与える影響を分析できる。

対象：2026年度 / 基準日 2026-09-02

0041 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：基礎

組込みソフトウェアでデバイスドライバ層を設ける主な目的はどれか。

ア．周辺レジスタや割込みなどハードウェア固有処理を隠蔽し、上位層へ一定のAPIを提供する。
イ．全アプリケーションをISRとして実装する。
ウ．ハードウェア仕様を無視する。
エ．全周辺機器を同じレジスタアドレスにする。

答え・解説 正答：ア

ア：ドライバ層でハードウェア依存性を局所化すると移植性・保守性を高めやすい。
イ：ドライバ抽象化の目的ではない。
ウ：むしろ仕様に基づいて制御する層である。
エ：物理仕様上不可能であり抽象化とも異なる。
総合解説：ドライバ層でハードウェア依存性を局所化すると移植性・保守性を高めやすい。 この問題では「ドライバ抽象化」として、デバイスドライバがハードウェア固有処理を上位ソフトウェアから抽象化する役割を理解できる。

対象：2026年度 / 基準日 2026-09-02

0042 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：基礎

メモリマップドI/Oのステータスレジスタをcから読み取るとき、通常のRAM変数と異なりコンパイラ最適化へ注意が必要な理由はどれか。

ア．I/Oレジスタは必ず不揮発メモリだから。
イ．ハードウェアがソフトウェアとは独立に値を変更するため、読み出しを省略・統合されると実状態を取得できない可能性がある。
ウ．全I/Oレジスタは読み出し禁止だから。
エ．volatileを付けると必ず排他制御も自動実装されるから。

答え・解説 正答：イ

ア：volatileの意味は不揮発性ではない。
イ：I/Oレジスタは外部要因で変化するため、volatile等の適切な宣言とアクセス規則が必要になる。
ウ：ステータス読出し可能なレジスタは多数存在する。
エ：volatileは排他や原子性を保証しない。
総合解説：I/Oレジスタは外部要因で変化するため、volatile等の適切な宣言とアクセス規則が必要になる。この問題では「MMIOとvolatile」として、Memory-mapped I/Oレジスタアクセスでvolatile相当の扱いが必要な理由を理解できる。

対象：2026年度 / 基準日 2026-09-02

0043 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：基礎

CMSIS-Driverの目的として最も適切なものはどれか。

ア．特定メーカーの1機種だけでしか動かないAPIを固定する。

イ．RTOSスケジューラを完全に置換する。

ウ．周辺ドライバの共通APIを定義し、ミドルウェアやアプリケーションからハードウェア依存を減らす。

エ．全周辺機器をソフトウェアエミュレーションへ変える。

答え・解説 正答：ウ

ア：共通化の目的に反する。

イ：ドライバAPIでありRTOSカーネルではない。

ウ：CMSIS-DriverはRTOSに依存しない汎用ドライバインタフェースを定義し、再利用と統合を容易にする。

エ：そのような目的ではない。

総合解説：CMSIS-DriverはRTOSに依存しない汎用ドライバインタフェースを定義し、再利用と統合を容易にする。 この問題では「共通Driver API」として、CMSIS-Driverのような共通APIが移植性とミドルウェア再利用に寄与することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0044 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：標準

低頻度で不定期に発生するUART受信を待つとき、CPUを他処理へ使いたい。一般に適した方式はどれか。

ア．最高優先度タスクで常時ステータスレジスタを読み続ける。

イ．受信状態を一切確認しない。

ウ．UARTを毎回リセットするだけ。

エ．受信割込みでイベントを検出し、必要ならタスクへ通知する。

答え・解説 正答：エ

ア：CPUを占有し他処理を妨げる。

イ：データを処理できない。

ウ：受信待機方法にならない。

エ：低頻度イベントでは常時ポーリングよりCPUを効率的に利用しやすい。

総合解説：低頻度イベントでは常時ポーリングよりCPUを効率的に利用しやすい。 この問題では「Polling vs Interrupt」として、ポーリングと割込み方式をイベント頻度とCPU負荷から選択できる。

対象：2026年度 / 基準日 2026-09-02

0045 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：標準

高速ADCから1KBブロックを周期的にRAMへ転送する。CPU負荷を抑えたい。適した方法はどれか。

ア．DMAで周辺からRAMへ転送し、ブロック完了時に割込みでタスクへ通知する。

イ．CPUが最高優先度で全サンプルをポーリングコピーする。

ウ．ADCデータを保存せず破棄する。

エ．DMA転送中ずっと全割込みを禁止する。

答え・解説 正答：ア

ア：DMAに転送を任せればCPUによる1ワードずつのコピーを減らせる。

イ：CPU負荷が大きい。

ウ：機能を満たさない。

エ：リアルタイム応答を悪化させる。

総合解説：DMAに転送を任せればCPUによる1ワードずつのコピーを減らせる。 この問題では「DMAドライバ設計」として、DMAを用いて大量I/O転送時のCPU介入を減らせる。

対象：2026年度 / 基準日 2026-09-02

0046 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：標準

UARTドライバ初期化で起動直後から不要な割込みが発生する。改善として適切な順序はどれか。

ア．最初に全割込みを有効化し、その後でメモリを初期化する。

イ．クロック・GPIO・UARTレジスタ・バッファを設定して割込み要因をクリアした後、最後に必要な割込みを有効化する。

ウ．UARTクロックを無効のまま送信する。

エ．初期化処理を毎バイト繰り返す。

答え・解説 正答：イ

ア：初期化途中の割込みで不定動作し得る。

イ：状態が整う前に割込みを有効化すると未初期化データをISRが参照する危険がある。

ウ：周辺が動作しない。

エ：不要なオーバヘッドとなる。

総合解説：状態が整う前に割込みを有効化すると未初期化データをISRが参照する危険がある。 この問題では「Driver初期化」として、ドライバ初期化順序でクロック・ピン・周辺設定・割込み有効化の依存関係を考慮できる。

対象：2026年度 / 基準日 2026-09-02

0047 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：標準

SPI転送に2ms掛かるが、その間CPUを別処理に使いたい。ドライバ設計として適切なものはどれか。

ア．2msの間、最高優先度で空ループする。
イ．割込みとDMAを無効化し、処理を停止する。
ウ．非同期転送を開始し、DMA/割込み完了時にコールバックやRTOS通知で待機タスクを起床する。
エ．転送完了確認を行わない。

答え・解説 正答：ウ

ア：CPUを浪費する。
イ：非同期化の目的に反する。
ウ：転送待ちをBusy Loopにせずイベント駆動にするとCPU時間を有効利用できる。
エ：データ整合性を保証できない。
総合解説：転送待ちをBusy Loopにせずイベント駆動にするとCPU時間を有効利用できる。 この問題では「非同期I/O」として、非同期ドライバで完了コールバックや通知を用いてタスクをブロック待ちにできる。

対象：2026年度 / 基準日 2026-09-02

0048 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：標準

同じ制御レジスタをタスクとISRがRead-Modify-Writeで更新する。保護がない場合に起こり得る問題はどれか。

ア．レジスタ容量が自動的に2倍になる。
イ．ISRが必ず削除される。
ウ．CPUクロックが0Hzになる。
エ．片方が古い値を基に書き戻し、もう片方の更新ビットを失う。

答え・解説 正答：エ

ア：競合とは無関係である。
イ：そのような動作はない。
ウ：直接的な結果ではない。
エ：RMWが原子的でないと同時更新の一部がLost Updateになる可能性がある。
総合解説：RMWが原子的でないと同時更新の一部がLost Updateになる可能性がある。 この問題では「レジスタRMW競合」として、Read-Modify-Writeで他タスクやISRの更新を失う競合を理解できる。

対象：2026年度 / 基準日 2026-09-02

0049 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：応用

I2CドライバがBUSY、TIMEOUT、BUS_ERRORなどの状態を返せる。上位層の設計として最も適切なものはどれか。

ア．エラー種別を区別し、再試行・バス回復・上位通知・安全状態移行など適切な処置を決める。

イ．全エラーを成功として無視する。

ウ．エラーが1回出たら必ず製品を永久停止する。

エ．戻り値を読まず毎回同じ処理を続ける。

答え・解説 正答：ア

ア：ドライバエラーを単に成功扱いせず、原因に応じた回復方針を設計することが重要である。

イ：データ破損や故障隠蔽につながる。

ウ：一時的BUSY等まで同一扱いするのは不適切である。

エ：異常検出できない。

総合解説：ドライバエラーを単に成功扱いせず、原因に応じた回復方針を設計することが重要である。 この問題では「Driverエラー処理」として、ドライバの戻り値・状態コードを上位層が処理する必要性を理解できる。

対象：2026年度 / 基準日 2026-09-02

0050 3-2 割込み・ドライバ > デバイスドライバ・I/O制御 ｜ 難易度：応用

センサIC変更のたびに制御アルゴリズムまで大幅修正が必要になっている。改善方針として最も適切なものはどれか。

ア．全レジスタアドレスを制御アルゴリズム内へ直接埋め込む。

イ．センサ固有のレジスタ操作をドライバへ隔離し、上位層には物理量取得など安定したインタフェースを提供する。

ウ．ドライバ層を削除して全処理をISRへ置く。

エ．センサ変更を禁止する。

答え・解説 正答：イ

ア：依存がさらに強くなる。

イ：ハードウェア依存性をドライバ境界へ閉じ込めると上位制御の再利用・単体テストを行いやすい。

ウ：責務分離とテスト性を悪化させる。

エ：設計上の改善にならない。

総合解説：ハードウェア依存性をドライバ境界へ閉じ込めると上位制御の再利用・単体テストを行いやすい。 この問題では「Driver境界設計」として、ドライバ層でハードウェア依存コードと上位ロジックを分離しテスト容易性を高められる。

対象：2026年度 / 基準日 2026-09-02

0051 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：基礎

Cortex-Mで不正な命令実行や不正メモリアクセスが発生したときに利用される仕組みとして最も適切なものはどれか。

ア．PWMデューティ比
イ．UARTストップビット
ウ．UsageFault、BusFault、MemManage、HardFaultなどのFault例外。
エ．I2Cプルアップ

答え・解説 正答：ウ

ア：例外診断機構ではない。
イ：通信フレーム要素である。
ウ：Cortex-Mでは原因に応じたFault例外とステータスレジスタを用いて異常を診断できる。
エ：電気レベル形成用である。
総合解説：Cortex-Mでは原因に応じたFault例外とステータスレジスタを用いて異常を診断できる。 この問題では「Fault例外」として、Cortex-MのFault例外が不正メモリアクセスや不正命令などの診断に使えることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0052 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：基礎

再現性の低いHardFaultを解析するため、Faultハンドラで優先して記録すると有用な情報はどれか。

ア．画面の背景色だけ
イ．CPUの製造年月日だけ
ウ．全RAMを消去した後の値だけ
エ．Faultステータスレジスタ、例外発生時PC/LR、スタックポインタなど。

答え・解説 正答：エ

ア：故障位置の診断情報にならない。
イ：個体情報だけではFault原因を特定できない。
ウ：障害時情報を失う。
エ：Fault原因と発生位置を特定するため、例外時コンテキストとステータス情報の保存が有効である。
総合解説：Fault原因と発生位置を特定するため、例外時コンテキストとステータス情報の保存が有効である。 この問題では「Fault診断ログ」として、HardFault発生時にFaultステータスやスタックフレームを保存する診断が有効であることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0053 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：基礎

UART受信データを固定RAM領域で連続的に蓄積し、末尾まで到達したら先頭へ戻して使用したい。適した構造はどれか。

ア．リングバッファ
イ．単方向リンクリストを毎バイト動的確保する方式だけ
ウ．スタックを受信データで上書きする
エ．受信データを常に破棄する

答え・解説 正答：ア

ア：循環バッファは固定配列とRead/Write位置を用いて連続ストリームを効率的に格納できる。
イ：固定メモリ要求では断片化や確保時間の問題が起こり得る。
ウ：実行コンテキストを破壊する。
エ：蓄積要件を満たさない。
総合解説：循環バッファは固定配列とRead/Write位置を用いて連続ストリームを効率的に格納できる。 この問題では「リングバッファ」として、リングバッファが固定容量で連続データを効率よく保持できることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0054 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：標準

ログ用リングバッファが満杯になったとき、古いデータより最新状態を優先する要件である。適した方針はどれか。

ア．バッファ境界を越えて隣接メモリへ書き続ける。
イ．最古データを上書きする方式を採用し、Overflow回数も記録する。
ウ．Overflowを検出しない。
エ．スタック領域へ自動拡張する。

答え・解説 正答：イ

ア：メモリ破壊を起こす。
イ：最新情報優先ならOverwrite oldestが要件に合い、欠落の事実を監視情報として残すとよい。
ウ：データ欠落を診断できない。
エ：安全なバッファ管理にならない。
総合解説：最新情報優先ならOverwrite oldestが要件に合い、欠落の事実を監視情報として残すとよい。 この問題では「Buffer Overflow方針」として、バッファOverflow時の方針をデータ重要度から設計できる。

対象：2026年度 / 基準日 2026-09-02

0055 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：標準

FreeRTOS Message Bufferの使用上の注意として正しいものはどれか。

ア．無制限数のWriterが同期なしで安全に同時書込みできる。

イ．ISRからは一切利用できない。

ウ．基本実装は単一Writer・単一Readerを前提とし、複数Writer/Readerなら呼出しを直列化する必要がある。

エ．Message Bufferは固定長1バイトしか送れない。

答え・解説 正答：ウ

ア：基本前提に反する。

イ：FromISR版APIが用意されている。

ウ：Message BufferはStream Buffer上に構築され、複数Writer/Reader利用時はアプリ側で直列化が必要になる。

エ：格納可能範囲で可変長メッセージを扱える。

総合解説：Message BufferはStream Buffer上に構築され、複数Writer/Reader利用時はアプリ側で直列化が必要になる。 この問題では「Message Buffer」として、Message Bufferが可変長の離散メッセージを扱い、単一Writer/Reader前提があることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0056 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：標準

あるタスクが時々原因不明でクラッシュする。スタック不足を疑う場合の確認として適切なものはどれか。

ア．スタックサイズを無条件に1バイトにする。

イ．Stack Overflowは組込みRTOSでは起こらないと仮定する。

ウ．全タスクで同じスタック領域を無保護共有する。

エ．Stack Overflow検出機能を有効にし、High Water Markなどで最大使用量と余裕を測定する。

答え・解説 正答：エ

ア：不足を悪化させる。

イ：代表的な不安定動作原因である。

ウ：コンテキスト破壊を起こす。

エ：タスクごとに独立スタックを持つため、実運用時の最大使用量を確認して適切にサイズ設計する。

総合解説：タスクごとに独立スタックを持つため、実運用時の最大使用量を確認して適切にサイズ設計する。 この問題では「Task Stack診断」として、スタックOverflow検出機能とHigh Water Markによる余裕確認の必要性を理解できる。

対象：2026年度 / 基準日 2026-09-02

0057 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：標準

実行中にRTOSオブジェクトの動的生成を行う設計で、Heap不足時の異常を早期検出したい。適切な方策はどれか。

ア．割当APIの戻り値を確認し、Malloc Failed Hook等を利用して異常ログや安全処理へ移行する。

イ．動的確保は必ず成功すると仮定して戻り値を無視する。

ウ．Heap不足時に隣接メモリへ自動上書きする。

エ．失敗したら割込みを永久禁止するだけ。

答え・解説 正答：ア

ア：動的確保は失敗し得るため、NULLや失敗コードを必ず処理する必要がある。

イ：Heap枯渇時にNULL参照等へつながる。

ウ：メモリ破壊となる。

エ：回復や安全状態の設計にならない。

総合解説：動的確保は失敗し得るため、NULLや失敗コードを必ず処理する必要がある。 この問題では「Malloc失敗処理」として、動的メモリ確保失敗を検出して安全に処理できる。

対象：2026年度 / 基準日 2026-09-02

0058 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：標準

ProducerがConsumerより速く、Queueが満杯になることがある。安全な設計として最も適切なものはどれか。

ア．Queue容量を超えて隣接RAMへ書く。

イ．送信失敗やタイムアウトを検出し、データ重要度に応じて待機・廃棄・Backpressure・異常通知を選ぶ。

ウ．送信APIの戻り値を常に無視する。

エ．Consumerを削除する。

答え・解説 正答：イ

ア：メモリ破壊である。

イ：Queue Fullは起こり得る実行時状態として明示的に扱い、無制限書込みを避ける必要がある。

ウ：データ欠落を検出できない。

エ：問題を悪化させる。

総合解説：Queue Fullは起こり得る実行時状態として明示的に扱い、無制限書込みを避ける必要がある。 この問題では「Queue Overflow処理」として、Queue FullやTimeoutを正常な異常系として処理できる。

対象：2026年度 / 基準日 2026-09-02

0059 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：応用

制御機器で致命的Faultが発生した。再起動すれば復旧することがあるが、原因解析と安全確保も必要である。適切な処理はどれか。

ア．出力を最大値のまま保持して即リセットする。

イ．Fault情報を全消去して原因を記録しない。

ウ．最小限のFault情報を保全し、出力を安全状態にした後、監視設計に従ってリセット・再起動する。

エ．例外ハンドラで無限ループし安全出力も変更しない。

答え・解説 正答：ウ

ア：危険状態が残る可能性がある。
イ：再発解析が困難になる。
ウ：Fault時は危険出力を抑えつつ診断情報を残し、規定された回復手順を実行するのが望ましい。
エ：安全要求を満たさない可能性がある。
総合解説：Fault時は危険出力を抑えつつ診断情報を残し、規定された回復手順を実行するのが望ましい。 この問題では「Fault Recovery」として、異常時に診断情報を保存して安全状態へ移行する設計を評価できる。

対象：2026年度 / 基準日 2026-09-02

0060 3-2 割込み・ドライバ > 例外処理・バッファ・異常時処理 ｜ 難易度：応用

UARTから毎秒2000バイト到着し、Consumerタスクが高優先度処理の影響で最大50ms実行できない。ほかの余裕を無視した最小蓄積量は何バイト程度か。

ア．10バイト（到着量を10分の1で見積り）

イ．1000バイト（停止時間を10倍側で見積り）

ウ．0バイト（停止中の到着を無視）

エ．100バイト（2000B/s×0.05s）

答え・解説 正答：エ

ア：到着量を10分の1に見積もっている。
イ：単純必要量の10倍である。
ウ：停止中に到着するデータを保持できない。
エ：2000B/s×0.05s=100Bである。実設計ではジッタやBurst分の安全余裕を追加する。
総合解説：2000B/s×0.05s=100Bである。実設計ではジッタやBurst分の安全余裕を追加する。 この問題では「Buffer容量見積り」として、受信バッファ容量を到着率・最悪処理停止時間から見積もれる。

対象：2026年度 / 基準日 2026-09-02

0061 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：基礎

ハードリアルタイムシステムの特徴として最も適切なものはどれか。

ア．定められたデッドラインを守ること自体が正しさの一部であり、期限違反が重大な障害になり得る。
イ．平均応答が速ければ一部の期限違反は常に問題にならない。
ウ．処理時間を一切測定しない。
エ．デッドラインを設定しないことが必須である。

答え・解説 正答：ア

ア：ハードリアルタイムでは計算結果だけでなく期限内に結果を出すことが必須条件になる。
イ：これはソフトリアルタイム寄りの考え方である。
ウ：期限保証には時間評価が必要である。
エ：リアルタイム要件と矛盾する。
総合解説：ハードリアルタイムでは計算結果だけでなく期限内に結果を出すことが必須条件になる。 この問題では「Hard/Soft Real-Time」として、ハードリアルタイムとソフトリアルタイムの違いを期限違反の扱いから説明できる。

対象：2026年度 / 基準日 2026-09-02

0062 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：基礎

リアルタイムタスクのWCETとして最も適切な説明はどれか。

ア．多数回測定した平均時間だけ。
イ．対象条件下で想定される最悪ケースの実行時間。
ウ．最短実行時間。
エ．タスク生成から削除までの寿命。

答え・解説 正答：イ

ア：平均値は最悪時間を保証しない。
イ：WCETはWorst-Case Execution Timeで、デッドライン保証や応答時間解析に用いる。
ウ：WCETとは逆である。
エ：実行時間の定義ではない。
総合解説：WCETはWorst-Case Execution Timeで、デッドライン保証や応答時間解析に用いる。 この問題では「WCET」として、WCETが平均実行時間ではなく最悪条件の実行時間評価であることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0063 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：基礎

周期1msの制御タスクが、ある周期では予定時刻から10μs遅れ、別の周期では80μs遅れて開始する。この開始時刻のばらつきを表す用語として適切なものはどれか。

ア．メモリリーク
イ．デッドロック
ウ．ジッタ
エ．量子化誤差

答え・解説 正答：ウ

ア：未解放メモリが蓄積する現象である。
イ：資源待ちの循環で進行不能になる現象である。
ウ：リアルタイム処理では周期・開始・完了時刻などの時間的ばらつきをジッタとして扱う。
エ：A/D変換などの離散化誤差である。
総合解説：リアルタイム処理では周期・開始・完了時刻などの時間的ばらつきをジッタとして扱う。 この問題では「Timing Jitter」として、ジッタを周期・応答時刻のばらつきとして理解できる。

対象：2026年度 / 基準日 2026-09-02

0064 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

高優先度周期タスクの最悪応答時間を見積もる際、考慮すべき要素として最も適切なものはどれか。

ア．平均CPU温度だけ。
イ．ソースコード行数だけ。
ウ．タスク名だけ。
エ．自身のWCET、より高優先度処理の干渉、共有資源によるブロッキング、割込み等の遅延。

答え・解説 正答：エ

ア：時間解析の主要要素ではない。
イ：実行時間を直接保証しない。
ウ：時間要件と無関係である。
エ：タスク応答時間は単独実行時間だけでなくスケジューリング干渉とブロッキングを含めて評価する。
総合解説：タスク応答時間は単独実行時間だけでなくスケジューリング干渉とブロッキングを含めて評価する。 この問題では「応答時間構成」として、応答時間に実行時間だけでなく干渉・ブロッキング・割込み遅延が含まれることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0065 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

周期5ms・WCET1msのタスクA、周期10ms・WCET2msのタスクB、周期20ms・WCET2msのタスクCがある。利用率合計はいくらか。

ア．50%（1/5＋2/10＋2/20）

イ．30%（一部タスクだけを計上）

ウ．70%（合計を過大に換算）

エ．100%（CPU全時間を使用すると仮定）

答え・解説 正答：ア

ア：1/5=20%、2/10=20%、2/20=10%で合計50%である。

イ：タスクの一部しか加えていない。

ウ：計算結果と一致しない。

エ：全CPU時間を使用する組合せではない。

総合解説：1/5=20%、2/10=20%、2/20=10%で合計50%である。 この問題では「利用率計算」として、周期タスクのCPU利用率を計算できる。

対象：2026年度 / 基準日 2026-09-02

0066 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

周期タスクでvTaskDelay()よりxTaskDelayUntil()系が適する理由はどれか。

ア．xTaskDelayUntil()はCPUを常に100%使用するから。

イ．前回の基準時刻から絶対的な次回起床時刻を求めるため、処理時間の変動が周期へ累積しにくい。

ウ．xTaskDelayUntil()は割込みを永久禁止するから。

エ．vTaskDelay()はタスクを削除するから。

答え・解説 正答：イ

ア：待機中はBlocked状態になる。

イ：相対遅延は関数呼出し時点から待つため処理時間分が周期へ入りやすい。

ウ：そのような機能ではない。

エ：vTaskDelay()も単なる遅延待機である。

総合解説：相対遅延は関数呼出し時点から待つため処理時間分が周期へ入りやすい。 この問題では「絶対時刻周期制御」として、xTaskDelayUntil系が周期タスクの一定周波数実行に向く理由を説明できる。

対象：2026年度 / 基準日 2026-09-02

0067 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

周期10msのタスクが11ms掛かった周期が発生した。デッドラインが周期と同じ10msなら、最も適切な判断はどれか。

ア．平均が10ms未満ならこの超過は必ず無視できる。
イ．11msは10msより短いので問題ない。
ウ．そのジョブはデッドラインミスであり、WCET・干渉・処理量を再評価する必要がある。
エ．デッドラインは処理時間と無関係である。

答え・解説 正答：ウ

ア：ハードリアルタイムでは単発の期限違反も問題になり得る。
イ：大小関係が誤っている。
ウ：完了時刻が期限を超えているためリアルタイム要件違反である。
エ：期限内完了が要件である。
総合解説：完了時刻が期限を超えているためリアルタイム要件違反である。 この問題では「Deadline Miss」として、デッドラインミスを測定し周期タスクの余裕を評価できる。

対象：2026年度 / 基準日 2026-09-02

0068 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

周期5ms、10ms、40msの三つの独立周期タスクへRate Monotonicで優先度を付ける。最高優先度にするタスクはどれか。

ア．周期40msのタスク（最長周期を最高優先度）
イ．周期10msのタスク（中間周期を最高優先度）
ウ．全タスクを同一優先度にする（全タスク同一優先度）
エ．周期5msのタスク（最短周期を最高優先度）

答え・解説 正答：エ

ア：最も長周期なので最高優先度ではない。
イ：5msタスクより優先度は低くなる。
ウ：Rate Monotonicの固定優先度規則ではない。
エ：Rate Monotonicでは周期が短いタスクほど高優先度にする。
総合解説：Rate Monotonicでは周期が短いタスクほど高優先度にする。 この問題では「RM優先度」として、Rate Monotonicの優先度割当を周期から判断できる。

対象：2026年度 / 基準日 2026-09-02

0069 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

最長80μsの割込み禁止区間が存在し、対象ISR自体の入口処理等に20μs掛かる。他の干渉を無視したとき最悪開始遅延は少なくとも何μs程度になり得るか。

ア．100μs（割込み禁止待ち＋入口処理）

イ．20μs（入口処理だけ）

ウ．60μs（割込み禁止時間から一部を減算）

エ．0μs（割込み禁止の影響を無視）

答え・解説 正答：ア

ア：80μsのマスク待ちに20μsの入口等が加われば100μs程度になり得る。

イ：割込み禁止時間を無視している。

ウ：計算結果と一致しない。

エ：割込み禁止区間が存在するため0にはならない。

総合解説：80μsのマスク待ちに20μsの入口等が加われば100μs程度になり得る。 この問題では「Interrupt Latency Budget」として、割込み禁止時間が最悪割込み応答へ直接加算され得ることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0070 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：標準

10万回の実機測定で最大実行時間が800μsだったので、WCETを800μsと断定した。問題点として最も適切なものはどれか。

ア．測定回数が1回を超えると必ず不正になる。

イ．測定で全入力・キャッシュ状態・割込み干渉などの最悪条件を網羅した保証がなければ、未観測の長い経路が残る可能性がある。

ウ．実行時間測定はリアルタイム設計で禁止されている。

エ．キャッシュ状態は実行時間に影響しない。

答え・解説 正答：イ

ア：多数回測定自体は問題ではない。

イ：WCET保証には測定条件の網羅性や静的解析など、最悪条件を確実に含む根拠が必要である。

ウ：実測は重要な評価手段の一つである。

エ：ヒット・ミスで変動する。

総合解説：WCET保証には測定条件の網羅性や静的解析など、最悪条件を確実に含む根拠が必要である。 この問題では「WCET測定の限界」として、測定だけでは未観測の最悪経路を保証できないことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0071 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：応用

高優先度タスクのWCETは2ms、周期・デッドラインは5msだが、低優先度タスクが最大4ms Mutexを保持する可能性がある。設計上の重要な問題はどれか。

ア．高優先度なので低優先度タスクのMutexには絶対待たない。

イ．Mutex保持時間は応答時間へ影響しない。

ウ．共有資源ブロッキングだけで期限余裕を超える可能性があり、Critical Section短縮や優先度継承等を検討すべきである。

エ．周期を測らず問題なしとする。

答え・解説 正答：ウ

ア：資源を保持されていれば待つ。
イ：ブロッキング時間として影響する。
ウ：自身のWCETが短くても共有資源待ちが長ければ応答時間が5msを超え得る。
エ：期限保証ができない。
総合解説：自身のWCETが短くても共有資源待ちが長ければ応答時間が5msを超え得る。 この問題では「Blocking Time」として、共有資源のブロッキングを含めて期限余裕を評価できる。

対象：2026年度 / 基準日 2026-09-02

0072 3-3 リアルタイム性能・資源制約 > 応答時間・最悪実行時間・タイミング ｜ 難易度：応用

1msデッドラインの制御処理について、タスクWCET=500μs、最大割込み干渉=250μs、最大Mutexブロッキング=150μs、その他オーバヘッド=50μsと見積もった。単純合計の余裕は何μsか。

ア．950μs（使用時間950μsを余裕と解釈）

イ．100μs（残余を100μsと換算）

ウ．0μs（余裕なしと解釈）

エ．50μs（1,000-950μs）

答え・解説 正答：エ

ア：これは使用する時間であり余裕ではない。
イ：計算結果と一致しない。
ウ：単純合計では50μs残る。
エ：500+250+150+50=950μsで、1,000μsまでの余裕は50μsである。
総合解説：500+250+150+50=950μsで、1,000μsまでの余裕は50μsである。 この問題では「タイミング予算」として、周期・WCET・割込み干渉を統合してデッドライン余裕を評価できる。

対象：2026年度 / 基準日 2026-09-02

0073 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：基礎

安全性重視の組込みRTOSで、起動後にタスクやQueueを増減しない。静的メモリ確保を選ぶ利点として適切なものはどれか。

ア．必要メモリを設計時に確保でき、実行時Heap断片化や動的確保失敗の不確実性を減らせる。
イ．RAMを一切使用しなくなる。
ウ．スタックOverflowが物理的に起こらなくなる。
エ．CPUクロックが不要になる。

答え・解説 正答：ア

ア：FreeRTOSではタスクやQueue等を静的に作成でき、メモリ使用を事前に固定できる。
イ：静的確保でもRAMは使用する。
ウ：サイズ不足なら静的でもOverflowは起こり得る。
エ：メモリ確保方式とクロックは無関係である。
総合解説：FreeRTOSではタスクやQueue等を静的に作成でき、メモリ使用を事前に固定できる。 この問題では「Static Allocation」として、静的メモリ確保が実行時確保失敗や断片化を避けやすいことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0074 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：基礎

RTOSでタスク数を増やすとRAM使用量が増えやすい主な理由の一つはどれか。

ア．各タスクが必ず専用CPUコアを必要とするため。
イ．各タスクにコンテキスト保存用を含む独自スタック領域が必要になるため。
ウ．タスクごとに別の電源ICが必要だから。
エ．タスク数に応じてROMが自動的にRAMへ変換されるため。

答え・解説 正答：イ

ア：単一コアでも多数タスクを切替実行できる。
イ：FreeRTOSタスクはそれぞれスタックを持ち、タスク数とスタックサイズがRAM使用量へ直接影響する。
ウ：RTOSタスクと物理電源ICは対応しない。
エ：そのような動作はない。
総合解説：FreeRTOSタスクはそれぞれスタックを持ち、タスク数とスタックサイズがRAM使用量へ直接影響する。 この問題では「Task Stack RAM」として、各タスクが独自スタックを持つことがRAM使用量に影響することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0075 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：基礎

動的確保と解放を長期間繰り返した結果、空きメモリ総量は十分なのに大きな連続領域を確保できない。この現象として適切なものはどれか。

ア．Priority Inversion
イ．Clock Jitter
ウ．外部断片化
エ．Aliasing

答え・解説 正答：ウ

ア：タスク優先度とMutexの問題である。
イ：時間ばらつきの問題である。
ウ：空き領域が細かく分散すると総空き量が十分でも大きな連続ブロックを確保できないことがある。
エ：信号標本化の折返し現象である。
総合解説：空き領域が細かく分散すると総空き量が十分でも大きな連続ブロックを確保できないことがある。 この問題では「Heap Fragmentation」として、Heap断片化が長期運用の大きな連続領域確保を失敗させる可能性を理解できる。

対象：2026年度 / 基準日 2026-09-02

0076 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：標準

各タスクのスタックを安全に削減したい。最も適切な方法はどれか。

ア．全タスクのスタックを一律最小値へ変更する。
イ．Stack Overflow検出を無効にして容量を減らす。
ウ．再帰呼出しの深さを無制限にする。
エ．代表的な最悪負荷試験でHigh Water Mark等を測り、割込み・関数呼出し・局所変数を考慮した安全余裕を残す。

答え・解説 正答：エ

ア：タスクごとの使用量差を無視している。
イ：故障検出能力を失う。
ウ：必要スタック量が予測しにくくなる。
エ：実測最大使用量を確認しながら余裕を設けて決定するのが適切である。
総合解説：実測最大使用量を確認しながら余裕を設けて決定するのが適切である。 この問題では「Stack Sizing」として、Stack High Water Markを用いてスタック余裕を評価できる。

対象：2026年度 / 基準日 2026-09-02

0077 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：標準

一つのタスクのポインタ暴走が他タスクの重要データを破壊する影響を抑えたい。ハードウェア支援として有効なものはどれか。

ア．MPUでタスクがアクセスできるメモリ領域と権限を制限する。
イ．全RAMを常に読み書き実行可能にする。
ウ．全タスクで同じスタックを共有する。
エ．ポインタ検証を禁止する。

答え・解説 正答：ア

ア：Memory Protection Unitを用いると領域単位のアクセス権を設定し、不正アクセスをFaultとして検出できる。
イ：隔離効果がない。
ウ：破壊リスクが高まる。
エ：安全性を悪化させる。
総合解説：Memory Protection Unitを用いると領域単位のアクセス権を設定し、不正アクセスをFaultとして検出できる。 この問題では「MPU保護」として、MPUによってタスクの不正メモリアクセスを制限できることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0078 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：標準

同じ128バイトの通信オブジェクトを高速に確保・解放し、実行時間のばらつきと断片化を抑えたい。適した方式はどれか。

ア．任意サイズのmalloc/freeを無制限に繰り返すことだけを採用する。
イ．固定サイズブロックのMemory Poolを事前確保する。
ウ．スタックを通信オブジェクトの永続保存領域として無制限使用する。
エ．オブジェクトを使うたびに全RAMを初期化する。

答え・解説 正答：イ

ア：断片化や時間変動が増える可能性がある。
イ：固定ブロックPoolは確保・解放処理を単純化し、外部断片化を避けやすい。
ウ：寿命とスコープの問題がある。
エ：大きなオーバヘッドとなる。
総合解説：固定ブロックPoolは確保・解放処理を単純化し、外部断片化を避けやすい。 この問題では「Memory Pool」として、固定サイズMemory Poolが確保時間予測性と断片化低減に有利なことを判断できる。

対象：2026年度 / 基準日 2026-09-02

0079 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：標準

Producerが毎10msに1件生成し、Consumerが高優先度処理により最大100ms停止する可能性がある。停止中の新規データを全て保持する単純な最小Queue長は何件か。

ア．1件（1周期分だけ保持）
イ．5件（停止時間の半分だけ保持）
ウ．10件（100ms÷10ms）
エ．100件（必要最小量より大きく確保）

答え・解説 正答：ウ

ア：停止中に複数イベントが到着する。
イ：半分しか保持できない。
ウ：100ms÷10ms=10件分が停止中に到着する。実設計では境界条件と余裕を追加する。
エ：単純最小量より大きい。
総合解説：100ms÷10ms=10件分が停止中に到着する。実設計では境界条件と余裕を追加する。 この問題では「Queue Capacity」として、Queue容量を生産率・消費率・最大停止時間から設計できる。

対象：2026年度 / 基準日 2026-09-02

0080 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：標準

大容量ネットワークパケットをQueueへ毎回コピーしてCPU負荷とRAM帯域が高い。改善策として有効なものはどれか。

ア．同じバッファを複数タスクが無保護で書き換える。
イ．全パケットを破棄する。
ウ．コピー回数をさらに増やす。
エ．固定バッファPoolを用い、Queueではバッファポインタやハンドルだけを渡し、所有権を明確に管理する。

答え・解説 正答：エ

ア：競合とデータ破損が起こり得る。
イ：機能要件を満たさない。
ウ：性能問題を悪化させる。
エ：大容量データのコピーを減らせるが、二重解放や同時更新を防ぐ所有権規則が必要である。
総合解説：大容量データのコピーを減らせるが、二重解放や同時更新を防ぐ所有権規則が必要である。 この問題では「Zero-Copy Buffer」として、メモリコピーを減らすZero-copy設計の利点と所有権管理の必要性を理解できる。

対象：2026年度 / 基準日 2026-09-02

0081 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：応用

RAM 128KBの製品で、起動後にタスク・Queue・Timerをユーザ操作に応じて無制限生成する設計になっている。改善方針として最も適切なものはどれか。

ア．必要最大数を要件から定め、静的または上限付きPoolで確保し、生成失敗時の処理も設計する。

イ．RAM上限を無視して生成を続ける。

ウ．確保APIの戻り値を確認しない。

エ．全オブジェクトへ最大サイズを無条件割り当てる。

答え・解説 正答：ア

ア：資源上限を明確化するとメモリ使用量と故障モードを予測できる。

イ：Heap枯渇に至る。

ウ：失敗時に不定動作へつながる。

エ：RAM浪費である。

総合解説：資源上限を明確化するとメモリ使用量と故障モードを予測できる。 この問題では「Resource Budget」として、実行時の動的生成を減らしメモリ使用上限を事前確認できる。

対象：2026年度 / 基準日 2026-09-02

0082 3-3 リアルタイム性能・資源制約 > メモリ・資源制約 ｜ 難易度：応用

タスク5個のスタック合計20KB、静的Queue/Buffer 30KB、グローバルデータ10KB、RTOS Heap 40KBを確保する。その他を無視した合計RAM使用量は何KBか。

ア．60KB（一部メモリ領域だけを合算）

イ．100KB（20+30+10+40）

ウ．40KB（Heapだけを計上）

エ．120KB（単純合計へ20KB追加）

答え・解説 正答：イ

ア：一部の領域を加えていない。

イ：20+30+10+40=100KBである。実際にはカーネル管理領域やアラインメント等も加わる。

ウ：Heapだけを数えている。

エ：単純合計より20KB多い。

総合解説：20+30+10+40=100KBである。実際にはカーネル管理領域やアラインメント等も加わる。 この問題では「RAM Budget」として、タスク数・スタック・Queue・Heapを統合したRAM予算を作成できる。

対象：2026年度 / 基準日 2026-09-02

0083 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：基礎

FreeRTOSのTickless Idleを利用する主な省電力効果はどれか。

ア．全タスクを永久削除する。

イ．CPUクロックを常に最大化する。

ウ．実行可能なタスクがない期間に周期Tick割込みを抑止し、MCUをより長く低消費電力状態へ置ける。

エ．割込みを永久禁止する。

答え・解説 正答：ウ

ア：Tickless Idleの機能ではない。

イ：省電力と逆である。

ウ：通常のTickで頻繁に起床する損失を減らし、次のイベントまで深いSleepを利用しやすくする。

エ：外部イベント等で起床できなくなる。

総合解説：通常のTickで頻繁に起床する損失を減らし、次のイベントまで深いSleepを利用しやすくする。 この問題では「Tickless Idle」として、Tickless Idleが不要な周期Tickを停止して低電力状態滞在時間を延ばすことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0084 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：基礎

100ms後に完了するI/Oを待つタスクで、最もCPU効率のよい方式はどれか。

ア．100ms間ステータスを最高速度で読み続ける。

イ．全割込みを禁止して待つ。

ウ．I/O完了を確認しない。

エ．割込み・Semaphore・Notification等を待ってBlocked状態にする。

答え・解説 正答：エ

ア：不要なCPUサイクルと電力を消費する。

イ：応答性を悪化させる。

ウ：正しい同期ができない。

エ：待ち時間にCPUを消費せず、他タスクや低電力状態へCPUを譲れる。

総合解説：待ち時間にCPUを消費せず、他タスクや低電力状態へCPUを譲れる。 この問題では「Busy Wait削減」として、Busy WaitよりBlocked待ちがCPU利用率と消費電力の低減に有効であることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0085 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：基礎

大量ADCデータを毎周期RAMへコピーするCPU負荷が高い。改善として最も適切なものはどれか。

ア．DMAへ転送をオフロードし、CPUはブロック単位の処理へ集中する。
イ．CPUで1バイトずつBusy Loop転送する。
ウ．ADCを停止してデータを捨てる。
エ．全タスクを最高優先度にする。

答え・解説 正答：ア

ア：専用DMAでメモリ転送を行うとCPU命令数を減らし、性能・電力を改善できる場合がある。
イ：CPU負荷が増える。
ウ：機能を満たさない。
エ：転送量は減らない。
総合解説：専用DMAでメモリ転送を行うとCPU命令数を減らし、性能・電力を改善できる場合がある。 この問題では「DMA Offload」として、DMAでCPUコピー処理を削減して性能と電力を改善できる。

対象：2026年度 / 基準日 2026-09-02

0086 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：標準

配列検索処理がCPU時間の40%を占めている。線形探索O(n)で毎回1万要素を検索している。最適化の第一候補として適切な考え方はどれか。

ア．CPUクロックだけを無条件に最大化する。
イ．データ特性が許せばハッシュや索引、二分探索可能な構造などアルゴリズム自体を見直す。
ウ．検索結果を毎回捨てる。
エ．全処理をCritical Sectionに入れる。

答え・解説 正答：イ

ア：電力・熱を増やし、根本的な計算量は変わらない。
イ：計算量を削減できれば、単純なクロック向上より大きな性能・電力改善につながる場合がある。
ウ：機能を満たさない。
エ：並行性と応答性を悪化させる。
総合解説：計算量を削減できれば、単純なクロック向上より大きな性能・電力改善につながる場合がある。 この問題では「Algorithm Optimization」として、アルゴリズム改善がクロック増加より根本的な性能改善になる場合を判断できる。

対象：2026年度 / 基準日 2026-09-02

0087 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：標準

システム高速化を行うときの手順として最も適切なものはどれか。

ア．測定せず全関数を同じように書き換える。

イ．最適化前後の比較を行わない。

ウ．実行時間・CPU使用率・コンテキストスイッチ・キャッシュミス等を計測してボトルネックを特定し、効果を再測定する。

エ．CPU使用率が高い関数を必ず削除する。

答え・解説 正答：ウ

ア：効果の小さい部分に工数を使う可能性がある。
イ：効果を検証できない。
ウ：測定に基づいて支配的な処理を改善し、前後比較することで最適化効果を確認できる。
エ：必要機能を失う可能性がある。
総合解説：測定に基づいて支配的な処理を改善し、前後比較することで最適化効果を確認できる。 この問題では「Performance Profiling」として、プロファイリングに基づいて実際のボトルネックを最適化できる。

対象：2026年度 / 基準日 2026-09-02

0088 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：標準

1件ごとに無線送信すると起床・送信オーバヘッドが大きい。10件まとめて送る方式へ変更した場合の一般的なトレードオフはどれか。

ア．省電力化と遅延0を必ず同時に達成できる。

イ．まとめるほどRAM使用量は必ず0になる。

ウ．Batchingするとデータ順序が必ず破壊される。

エ．起床・通信回数を減らして省電力化しやすい一方、個々のデータ送信遅延は増える。

答え・解説 正答：エ

ア：一般にはトレードオフがある。
イ：バッファ容量が必要になる。
ウ：適切に設計すれば順序は保持できる。
エ：Batchingは固定オーバヘッドを共有できるが、バッファに貯める待ち時間が加わる。
総合解説：Batchingは固定オーバヘッドを共有できるが、バッファに貯める待ち時間が加わる。 この問題では「Batch Processing」として、処理をまとめて行うBatchingが起床回数削減とレイテンシ増加のトレードオフを持つことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0089 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：応用

CPU周波数を半分に下げて省電力化したい。リアルタイムタスクがある場合に最初に確認すべきことはどれか。

ア．周波数低下後のWCETと最悪応答時間を再評価し、全デッドラインを満たせるか確認する。

イ．周波数を下げても実行時間は必ず同じと仮定する。

ウ．デッドラインを削除する。

エ．全割込みを無効化する。

答え・解説 正答：ア

ア：省電力化で演算時間が延びるため、リアルタイム余裕を維持できる範囲で周波数を設定する必要がある。

イ：CPU依存処理は一般に長くなる。

ウ：要件変更なしには不適切である。

エ：リアルタイム応答を失う。

総合解説：省電力化で演算時間が延びるため、リアルタイム余裕を維持できる範囲で周波数を設定する必要がある。 この問題では「DVFSとDeadline」として、DVFS等の性能・電力制御でデッドライン余裕を確認する必要がある。

対象：2026年度 / 基準日 2026-09-02

0090 3-3 リアルタイム性能・資源制約 > 性能・消費電力最適化 ｜ 難易度：応用

キャッシュを有効にすると平均処理時間は30%短縮するが、ミス時の実行時間ばらつきが増える。ハードリアルタイム制御への適用判断として最も適切なものはどれか。

ア．平均が速いのでWCETは確認しない。

イ．平均性能、WCET、ジッタ、消費電力を測定し、期限保証できる範囲でキャッシュやTCM配置を使い分ける。

ウ．キャッシュを有効にすれば必ず全アクセス時間が一定になる。

エ．消費電力とタイミングは性能設計に無関係である。

答え・解説 正答：イ

ア：期限違反リスクが残る。

イ：ハードリアルタイムでは平均高速化だけでなく最悪時間の決定性を評価する必要がある。

ウ：ヒット・ミスで変動する。

エ：組込みでは重要な制約である。

総合解説：ハードリアルタイムでは平均高速化だけでなく最悪時間の決定性を評価する必要がある。 この問題では「性能・電力トレードオフ」として、性能最適化で平均値だけでなく最悪値と消費電力を同時に評価できる。

対象：2026年度 / 基準日 2026-09-02

0091 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：基礎

保守性の高いモジュール設計として、一般に望ましい組合せはどれか。

ア．各モジュールの責務を明確にして凝集度を高め、モジュール間依存を必要最小限にして結合度を低くする。

イ．凝集度を低くし、全モジュールが互いの内部データへ直接アクセスする。

ウ．全機能を一つの巨大関数へ集約する。

エ．モジュール間のインタフェース仕様を定義しない。

答え・解説 正答：ア

ア：責務を局所化し依存を減らすと、変更影響範囲を限定しやすく、単体テストや再利用もしやすい。

イ：責務が分散し依存が強くなるため保守性が低下する。

ウ：変更・テスト・再利用の単位が大きくなり、責務分離が困難になる。

エ：依存関係が曖昧になり、統合時の不具合が増えやすい。

総合解説：責務を局所化し依存を減らすと、変更影響範囲を限定しやすく、単体テストや再利用もしやすい。 この問題では「凝集度と結合度」として、モジュールの凝集度を高くし結合度を低くする設計原則を説明できる。

対象：2026年度 / 基準日 2026-09-02

0092 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：基礎

情報隠蔽の考え方として最も適切なものはどれか。

ア．全ての内部変数をグローバル変数として公開する。

イ．モジュール内部のデータ構造や実装詳細を外部へ直接公開せず、定義したインタフェース経由で利用させる。

ウ．関数名を短くすることだけを意味する。

エ．実装コードを暗号化することを意味する。

答え・解説 正答：イ

ア：外部依存が増え、変更影響が広がる。

イ：内部実装を隠蔽すると、外部利用側を変更せずに内部を差し替えやすくなる。

ウ：情報隠蔽は命名規則ではなく、公開範囲と依存関係の設計原則である。

エ：ソフトウェア設計上のカプセル化と暗号化は別概念である。

総合解説：内部実装を隠蔽すると、外部利用側を変更せずに内部を差し替えやすくなる。 この問題では「情報隠蔽」として、情報隠蔽によってモジュール内部実装の変更影響を抑えられることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0093 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：基礎

UMLクラス図で、あるクラスが別のクラスの属性や操作を継承する関係を表すものはどれか。

ア．依存（Dependency）だけ
イ．ノート（Comment）
ウ．汎化（Generalization）
エ．パッケージインポートだけ

答え・解説 正答：ウ

ア：依存はある要素の変更が他へ影響し得る関係で、継承そのものではない。
イ：注釈を付ける要素で継承関係ではない。
ウ：UMLの汎化は、より具体的な分類子が一般的な分類子の特性を継承する関係を表す。
エ：名前空間利用の関係でクラス継承を表すものではない。
総合解説：UMLの汎化は、より具体的な分類子が一般的な分類子の特性を継承する関係を表す。 この問題では「クラス図の継承」として、UMLクラス図における継承関係を理解できる。

対象：2026年度 / 基準日 2026-09-02

0094 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：標準

通信装置の動作が「初期化」「接続待ち」「通信中」「エラー」の状態によって許可されるイベントが異なる。適した設計表現はどれか。

ア．全状態を一つの無条件ループとして表し、遷移条件を書かない。
イ．クラス名だけを一覧化する。
ウ．CPUクロック周波数だけを記述する。
エ．状態機械図で状態、イベント、遷移条件、アクションを明示する。

答え・解説 正答：エ

ア：状態依存性が見えず設計検証が困難になる。
イ：状態遷移の条件やイベントを表現できない。
ウ：振る舞い設計にはならない。
エ：状態依存動作を状態機械として表現すると、遷移漏れや不正遷移をレビューしやすい。
総合解説：状態依存動作を状態機械として表現すると、遷移漏れや不正遷移をレビューしやすい。 この問題では「状態機械設計」として、状態遷移を状態機械としてモデル化する利点を判断できる。

対象：2026年度 / 基準日 2026-09-02

0095 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：標準

一つのモジュールがセンサ読取、通信、ログ保存、UI表示を全て担当し、どの変更でも同じコードを修正する必要がある。改善として最も適切なものはどれか。

ア．機能責務ごとにモジュールを分離し、明確なインタフェースで連携させる。

イ．さらに機能を追加して一つのモジュールへ集約する。

ウ．全データを共有グローバル変数にする。

エ．インタフェース定義を削除する。

答え・解説 正答：ア

ア：変更理由の異なる責務を分離すると、変更影響とテスト範囲を限定しやすい。

イ：責務集中を悪化させる。

ウ：結合度を高め変更影響を広げる。

エ：モジュール境界が不明確になる。

総合解説：変更理由の異なる責務を分離すると、変更影響とテスト範囲を限定しやすい。 この問題では「責務分離」として、単一責任の考え方で変更理由を局所化できる。

対象：2026年度 / 基準日 2026-09-02

0096 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：標準

制御アルゴリズムが特定センサICのレジスタ操作を直接呼び出しており、センサ変更のたびに制御ロジックを修正している。改善として適切なものはどれか。

ア．制御アルゴリズム内へさらに多くのレジスタアドレスを埋め込む。

イ．温度取得などの抽象インタフェースを定義し、具体的なセンサドライバをその実装として差し替える。

ウ．センサICを変更できないようにする。

エ．全センサ処理を割込みハンドラへ移す。

答え・解説 正答：イ

ア：デバイス依存が強くなる。

イ：上位ロジックを具体デバイスから分離すると、ハードウェア変更への耐性とテスト容易性が高まる。

ウ：設計改善ではない。

エ：依存分離とは無関係である。

総合解説：上位ロジックを具体デバイスから分離すると、ハードウェア変更への耐性とテスト容易性が高まる。 この問題では「抽象インタフェース」として、依存性逆転に近い設計で上位ロジックを具体的デバイスから分離できる。

対象：2026年度 / 基準日 2026-09-02

0097 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：標準

モジュールAがBを呼び、BがCを呼び、CがAを直接呼び戻す構造になっている。主な問題はどれか。

ア．必ず実行速度が無限大になる。
イ．循環依存があるとRAMが不揮発化する。
ウ．循環依存により独立したビルド・テスト・再利用や変更影響の把握が難しくなる。
エ．循環依存は必ず正しいオブジェクト指向設計を意味する。

答え・解説 正答：ウ

ア：循環依存と実行速度にはそのような関係はない。
イ：メモリ特性とは無関係である。
ウ：モジュール間の依存方向が循環すると分離性が低下し、保守性を損ねやすい。
エ：循環依存はむしろ設計上の警戒点である。
総合解説：モジュール間の依存方向が循環すると分離性が低下し、保守性を損ねやすい。 この問題では「循環依存」として、循環依存がモジュール再利用と単体テストを困難にすることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0098 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：標準

異なる通信方式UART、SPI、CANを同じ上位送信APIから利用したいが、各実装の内部処理は大きく異なる。適切な設計はどれか。

ア．全通信方式の内部変数を一つの巨大構造体へ混在させる。
イ．通信方式ごとに上位アプリ全体を複製する。
ウ．共通APIを設けず、上位層が各レジスタを直接操作する。
エ．共通インタフェースを定義し、各通信ドライバへ処理を委譲する構成を検討する。

答え・解説 正答：エ

ア：結合度が高まり変更影響が広がる。
イ：重複コードが増え保守性が低下する。
ウ：ハードウェア依存性が上位へ漏れる。
エ：上位層から利用方法を統一しつつ、各方式固有の実装を分離できる。
総合解説：上位層から利用方法を統一しつつ、各方式固有の実装を分離できる。 この問題では「継承と合成」として、継承より委譲・合成が適する場合を判断できる。

対象：2026年度 / 基準日 2026-09-02

0099 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：応用

通信プロトコルは毎年変更されるが、安全監視ロジックは認証済みで頻繁に変更したくない。設計方針として最も適切なものはどれか。

ア．通信層と安全監視層を明確に分離し、安定した契約インタフェースを介して連携させる。
イ．両機能を同じ巨大関数に混在させる。
ウ．通信変更のたびに安全監視も無条件で全面書換えする。
エ．インタフェース仕様を管理しない。

答え・解説 正答：ア

ア：変更頻度や品質保証要求の異なる領域を分離すると、認証済み部分への変更波及を抑えられる。
イ：通信変更が安全監視へ波及しやすくなる。
ウ：不要な変更リスクが増える。
エ：変更影響を制御できなくなる。
総合解説：変更頻度や品質保証要求の異なる領域を分離すると、認証済み部分への変更波及を抑えられる。この問題では「変更容易性設計」として、変更頻度の異なる機能を分離して保守性を高められる。

対象：2026年度 / 基準日 2026-09-02

0100 3-4 ソフトウェア設計・開発手法 > 構造化・オブジェクト指向・モジュール化 ｜ 難易度：応用

制御モジュールの単体テストで実センサが必須になっており、自動テストが困難である。改善として最も適切なものはどれか。

ア．制御モジュール内部から直接物理アドレスへアクセスし続ける。
イ．センサアクセスを抽象インタフェース経由にし、テスト時はStub/Mock実装へ差し替えられるようにする。
ウ．テストを廃止して実機だけで確認する。
エ．実センサがない場合は常にテスト成功扱いする。

答え・解説 正答：イ

ア：テスト環境との分離が困難になる。
イ：依存先を差し替え可能にすると、実ハードウェアなしで入力条件や異常応答を再現できる。
ウ：早期欠陥検出と再現性を失う。
エ：品質を検証できない。
総合解説：依存先を差し替え可能にすると、実ハードウェアなしで入力条件や異常応答を再現できる。 この問題では「テスト容易なモジュール化」として、モジュール境界をテストダブル利用可能な設計にして単体テスト性を高められる。

対象：2026年度 / 基準日 2026-09-02

0101 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：基礎

モデルベース開発の基本的な考え方として最も適切なものはどれか。

ア．モデルを作成したら実装やテストは一切不要になる。

イ．モデルは画面装飾だけを目的とする。

ウ．システムや制御ロジックをモデルで表現し、設計・シミュレーション・検証・コード生成などへ活用する。

エ．モデルベース開発では要求仕様を扱わない。

答え・解説 正答：ウ

ア：モデルと実機の差異や生成コードの検証は依然必要である。

イ：設計・解析・検証のための工学的表現として用いられる。

ウ：モデルを中心成果物として利用することで早期検証や設計意図の共有に役立つ。

エ：要求からモデルへトレーサビリティを持たせることが重要である。

総合解説：モデルを中心成果物として利用することで早期検証や設計意図の共有に役立つ。 この問題では「モデルベース開発」として、モデルを要求・設計・検証の共通表現として使うモデルベース開発の基本を理解できる。

対象：2026年度 / 基準日 2026-09-02

0102 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：基礎

制御モデル自体をシミュレーション環境上で検証し、量産ECUハードウェアをまだ使用しない段階として最も適切なものはどれか。

ア．HIL（Hardware-in-the-Loop）

イ．量産検査だけ

ウ．フィールド保守だけ

エ．MIL（Model-in-the-Loop）

答え・解説 正答：エ

ア：HILでは実際のECU等のハードウェアをシミュレータへ接続して検証する。

イ：モデル段階の検証ではない。

ウ：開発初期のシミュレーションとは異なる。

エ：MILはモデル段階で制御対象モデルと制御モデルを組み合わせて振る舞いを検証する。

総合解説：MILはモデル段階で制御対象モデルと制御モデルを組み合わせて振る舞いを検証する。 この問題では「MIL/SIL/HIL」として、MIL・SIL・HILの対象の違いを理解できる。

対象：2026年度 / 基準日 2026-09-02

0103 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：基礎

Scrum Guide 2020におけるSprintの説明として最も適切なものはどれか。

ア．1か月以内の固定長イベントで、価値あるIncrementを作るための他のScrumイベントを包含する。

イ．要件が完全に固定されるまで開始できない。

ウ．Sprint中は検査や適応を行わない。

エ．必ず6か月以上の期間にする。

答え・解説 正答：ア

ア：SprintはScrumの中心的な反復単位であり、Sprint Planning、Daily Scrum、Review、Retrospectiveを含む。

イ：Scrumは経験主義に基づき反復的に適応する。

ウ：透明性・検査・適応がScrumの基本である。

エ：Scrum GuideではSprintは1か月以内である。

総合解説：SprintはScrumの中心的な反復単位であり、Sprint Planning、Daily Scrum、Review、Retrospectiveを含む。 この問題では「Scrum Sprint」として、ScrumのSprintが固定長イベントでありIncrementを生み出す単位であることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0104 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：標準

Scrumで、プロダクト改善に必要となり得る作業を優先順位付きで管理する単一の情報源はどれか。

ア．Sprint Backlogだけ

イ．Product Backlog

ウ．Definition of Doneだけ

エ．Daily Scrumだけ

答え・解説 正答：イ

ア：Sprint Backlogは選択したProduct Backlog ItemとSprint Goal達成計画である。

イ：Product Backlogはプロダクト改善に必要なものの創発的・順序付けされた一覧である。

ウ：完成基準であり作業一覧そのものではない。

エ：イベントであり成果物ではない。

総合解説：Product Backlogはプロダクト改善に必要なものの創発的・順序付けされた一覧である。 この問題では「Scrum Artifact」として、Product BacklogとSprint Backlogの役割を区別できる。

対象：2026年度 / 基準日 2026-09-02

0105 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：標準

新しい組込み製品でユーザー要求の不確実性が高い。開発初期から短い反復で動くIncrementを作る利点として最も適切なものはどれか。

ア．初回反復で全要求が永久固定される。

イ．テストを最後の1回だけに限定できる。

ウ．早期に実物に近い成果へフィードバックを得て、要求誤解や技術リスクを小さい単位で修正できる。

エ．設計変更が一切できなくなる。

答え・解説 正答：ウ

ア：反復開発では学習に応じてBacklogを更新し得る。
イ：各反復で品質確認を行うのが望ましい。
ウ：反復・漸増型は学習と適応を早め、大規模手戻りのリスクを下げやすい。
エ：むしろ変更へ段階的に対応しやすい。
総合解説：反復・漸増型は学習と適応を早め、大規模手戻りのリスクを下げやすい。 この問題では「反復・漸増開発」として、反復開発で早期フィードバックを得てリスクを低減できる。

対象：2026年度 / 基準日 2026-09-02

0106 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：標準

ISO/IEC/IEEE 12207:2026の適用について最も適切な説明はどれか。

ア．ウォーターフォール以外の方法を禁止する。

イ．組込みソフトウェアには適用できない。

ウ．保守や廃棄は対象外である。

エ．ソフトウェアライフサイクルの共通プロセス枠組みを示すが、特定の開発モデルや方法論を一律に強制するものではない。

答え・解説 正答：エ

ア：現行規格は特定モデルを必須としていない。
イ：ファームウェアを含む組込みシステムにも適用可能である。
ウ：ライフサイクル全体を扱う。
エ：ISO/IEC/IEEE 12207はライフサイクルプロセスを定義し、アジャイルを含むさまざまな工学アプローチへ適用できる。
総合解説：ISO/IEC/IEEE 12207はライフサイクルプロセスを定義し、アジャイルを含むさまざまな工学アプローチへ適用できる。 この問題では「ライフサイクルプロセス」として、ISO/IEC/IEEE 12207が特定のライフサイクルモデルを強制しないことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0107 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：標準

モデルベース開発で安全要求が変更された。設計管理として最も適切な方策はどれか。

ア．要求IDとモデル要素・テストケースの対応を管理し、変更の影響先を追跡できるようにする。
イ．要求とモデルの対応関係を記録しない。
ウ．変更された要求だけを削除し、モデルはそのままにする。
エ．モデルからテストへの対応は一切不要とする。

答え・解説 正答：ア

ア：トレーサビリティにより要求変更が設計・実装・検証へ正しく反映されたか確認しやすくなる。
イ：変更漏れを検出しにくくなる。
ウ：要求不整合が残る。
エ：検証網羅性を確認しにくくなる。
総合解説：トレーサビリティにより要求変更が設計・実装・検証へ正しく反映されたか確認しやすくなる。この問題では「モデル・要求トレーサビリティ」として、モデルと要求のトレーサビリティを維持する必要性を判断できる。

対象：2026年度 / 基準日 2026-09-02

0108 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：標準

安全関連組込みソフトをScrumで開発する。Sprintが短いことを理由に安全レビューを省略しようとしている。最も適切な判断はどれか。

ア．Scrumではレビューやテストは禁止されている。
イ．開発方法にかかわらず必要な安全・品質活動を各Incrementやライフサイクルへ組み込み、Definition of Done等で明確化する。
ウ．Sprintが短ければ安全要求は適用されない。
エ．安全活動は製品出荷後だけ行う。

答え・解説 正答：イ

ア：Scrumは検査と適応を重視する。
イ：アジャイルは品質要求を免除するものではなく、必要な検証や文書化を反復へ統合する必要がある。
ウ：安全要求は開発周期の長さに依存しない。
エ：早期・継続的な品質保証が必要である。
総合解説：アジャイルは品質要求を免除するものではなく、必要な検証や文書化を反復へ統合する必要がある。 この問題では「アジャイルと品質保証」として、アジャイルでも品質基準や安全要求を満たす必要があることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0109 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：応用

制御モデルはMILでは正常だが、量産コード生成後や実ECUでのタイミング差を検出したい。適切な検証方針はどれか。

ア．MILが通れば以後の検証は全て不要とする。

イ．最初から市場実車だけで試す。

ウ．MILに加えてSILで実装コード相当を検証し、さらにHILで実ECUとリアルタイム環境を組み合わせて差異を確認する。

エ．モデルと実装の比較を行わない。

答え・解説 正答：ウ

ア：実装・ハードウェア固有の問題を見逃す。

イ：危険なコーナーケースの再現性と安全性が低い。

ウ：検証対象を段階的に実装へ近づけることで、モデル誤り、コード生成差、I/Oやタイミング問題を切り分けやすい。

エ：差異検出ができない。

総合解説：検証対象を段階的に実装へ近づけることで、モデル誤り、コード生成差、I/Oやタイミング問題を切り分けやすい。 この問題では「段階的シミュレーション検証」として、MILからSIL/HILへ段階的に検証対象を実装へ近づける意義を評価できる。

対象：2026年度 / 基準日 2026-09-02

0110 3-4 ソフトウェア設計・開発手法 > モデルベース・アジャイル・開発プロセス ｜ 難易度：応用

要求変動が大きいUI機能と、厳密な安全検証が必要なモータ停止機能を同一製品で開発する。最も適切な考え方はどれか。

ア．UIと安全機能を区別せず全て同じ文書量・レビュー深度へ固定する。

イ．安全機能だけテストを省略する。

ウ．要求変動がある機能は構成管理しない。

エ．製品全体へ一律の作業方法を機械的に適用せず、要求特性とリスクに応じてプロセスをテーラリングしつつ全体のトレーサビリティを保つ。

答え・解説 正答：エ

ア：リスクに応じた資源配分ができない。

イ：高リスク領域ほど検証が重要である。

ウ：変更追跡が必要である。

エ：ライフサイクルプロセスは目的に応じて適用でき、アジャイル性と高保証活動を組み合わせることも可能である。

総合解説：ライフサイクルプロセスは目的に応じて適用でき、アジャイル性と高保証活動を組み合わせることも可能である。 この問題では「プロセステーラリング」として、開発プロセスの選択を要求変動性・安全性・組織能力から判断できる。

対象：2026年度 / 基準日 2026-09-02

0111 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：基礎

再利用可能なドライバAPI仕様に最低限含めるべき情報として最も適切なものはどれか。

ア．引数の意味と範囲、戻り値・エラー、呼出し前提、同期/非同期動作、再入可能性など。

イ．関数名だけ。

ウ．実装者の個人的なメモだけ。

エ．内部レジスタ配置だけ。

答え・解説 正答：ア

ア：利用側が正しく呼び出すための契約条件を明確にすると、統合時の誤解を減らせる。

イ：利用条件やエラー処理を判断できない。

ウ：利用者向け契約として不十分である。

エ：上位APIの振る舞い契約を説明できない。

総合解説：利用側が正しく呼び出すための契約条件を明確にすると、統合時の誤解を減らせる。 この問題では「API契約」として、API契約に入力・出力・エラー・前提条件を明示する必要性を理解できる。

対象：2026年度 / 基準日 2026-09-02

0112 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：基礎

FreeRTOS Queueで通常のデータ項目を送信した場合の基本的な特徴はどれか。

ア．送信側のローカル変数へのポインタだけを必ず保存する。

イ．Queue内部へ項目内容がコピーされるため、送信側の元変数が後で変化してもQueue内データは独立して保持される。

ウ．Queueは1バイトしか送れない。

エ．Queueはタスク間通信に使えない。

答え・解説 正答：イ

ア：通常のQueue項目は値をコピーして格納する。

イ：FreeRTOS Queueは項目をコピーして格納するため、データ所有期間を分離しやすい。

ウ：作成時に項目サイズを指定できる。

エ：代表的なタスク間通信機構である。

総合解説：FreeRTOS Queueは項目をコピーして格納するため、データ所有期間を分離しやすい。 この問題では「Queueによるメッセージ受渡し」として、Queueがコピー渡しで送受信側のデータ寿命を分離できることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0113 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：基礎

一つのISRから一つのタスクへ処理開始だけを通知したい。データ列の保持は不要である。最も軽量な候補はどれか。

ア．必ず1000要素Queueを使う。
イ．MutexをISRから取得する。
ウ．Direct-to-task notification
エ．グローバルフラグを無保護でBusy Pollingする。

答え・解説 正答：ウ

ア：要件に対して過剰である。
イ：ISRではMutexを使うべきでない。
ウ：特定タスクへの軽量なイベント通知として利用でき、Binary/Counting Semaphore相当の用途にも使える。
エ：競合やCPU浪費の原因になる。
総合解説：特定タスクへの軽量なイベント通知として利用でき、Binary/Counting Semaphore相当の用途にも使える。 この問題では「Task Notification再利用」として、Task Notificationが単一受信タスクへの軽量通知に適することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0114 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：標準

公開APIの引数の意味を変更したが関数名は同じままである。既存利用側への影響として最も適切なものはどれか。

ア．関数名が同じなら必ず完全互換である。
イ．引数の意味変更はテスト不要である。
ウ．API仕様書は更新しなくてよい。
エ．ソース上はビルドできても意味的互換性を失う可能性があり、仕様変更として影響分析と移行対応が必要である。

答え・解説 正答：エ

ア：意味変更による不具合が起こり得る。
イ：既存呼出しへの影響を確認する必要がある。
ウ：契約と実装の不一致を招く。
エ：API互換性は名前や型だけでなく、振る舞い契約の互換性も含めて評価する。
総合解説：API互換性は名前や型だけでなく、振る舞い契約の互換性も含めて評価する。 この問題では「API互換性」として、インタフェース仕様変更で後方互換性を評価できる。

対象：2026年度 / 基準日 2026-09-02

0115 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：標準

二つのタスクが同じ構造体を共有メモリで読み書きする。安全な設計として最も適切なものはどれか。

ア．書込み主体、読出しタイミング、排他/原子性、データ有効状態を明確に定義する。
イ．両タスクが任意時刻に無保護で全フィールドを書き換える。
ウ．volatileを付ければ自動的にMutex相当の排他が保証される。
エ．共有メモリでは同期設計が不要である。

答え・解説 正答：ア

ア：共有メモリは高速だが同期規則が曖昧だと競合や途中更新の読取りが発生する。
イ：競合と不整合の原因になる。
ウ：volatileは排他や原子性を保証しない。
エ：明確な同期方式が必要である。
総合解説：共有メモリは高速だが同期規則が曖昧だと競合や途中更新の読取りが発生する。 この問題では「共有メモリIPC」として、共有メモリ通信では排他・可視性・所有権を設計する必要がある。

対象：2026年度 / 基準日 2026-09-02

0116 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：標準

一つのProducerから一つのConsumerへ、長さが毎回異なるコマンドメッセージを送りたい。FreeRTOSで適した候補はどれか。

ア．Recursive Mutexだけ
イ．Message Buffer
ウ．Idleタスク
エ．ウォッチドッグ

答え・解説 正答：イ

ア：排他機構でありメッセージ保存機能ではない。
イ：Message Bufferは可変長の離散メッセージを格納でき、単一Writer/Reader用途に適する。
ウ：通信バッファではない。
エ：時間監視機能である。
総合解説：Message Bufferは可変長の離散メッセージを格納でき、単一Writer/Reader用途に適する。 この問題では「Message Buffer再利用」として、Message Bufferを可変長メッセージ通信へ利用できる。

対象：2026年度 / 基準日 2026-09-02

0117 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：標準

他製品でも使う通信コンポーネントをライブラリ化する。再利用性を高める方策として適切なものはどれか。

ア．暗黙の前提を文書化せず実装者だけが知るようにする。

イ．ハードウェア依存処理を上位APIへ露出させる。

ウ．必要OS機能、メモリ量、割込み条件、設定パラメータ、API契約、バージョン互換性を明示する。

エ．バージョン情報を付けない。

答え・解説 正答：ウ

ア：他製品で誤用されやすい。
イ：移植性が低下する。
ウ：再利用側が適用可否を判断できるよう依存条件と制約を明確にする必要がある。
エ：互換性管理が困難になる。
総合解説：再利用側が適用可否を判断できるよう依存条件と制約を明確にする必要がある。 この問題では「再利用コンポーネント契約」として、再利用部品で依存関係・設定・前提条件を文書化する必要性を理解できる。

対象：2026年度 / 基準日 2026-09-02

0118 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：標準

大容量フレームをZero-copyでProducerからConsumerへ渡す。最も重要な設計事項はどれか。

ア．両者が同時に自由に書き換える。

イ．解放責任を定義しない。

ウ．バッファ寿命を呼出し側ローカル変数より短くする。

エ．バッファ所有権がいつProducerからConsumerへ移り、誰が解放・再利用できるかを明確にする。

答え・解説 正答：エ

ア：データ競合が発生する。
イ：リークや二重解放の原因になる。
ウ：参照先が無効になる危険がある。
エ：コピーを省く代わりに、同時書換えや二重解放を防ぐ所有権規則が必要になる。
総合解説：コピーを省く代わりに、同時書換えや二重解放を防ぐ所有権規則が必要になる。 この問題では「Zero-copy所有権」として、Zero-copyの所有権移譲を明確化できる。

対象：2026年度 / 基準日 2026-09-02

0119 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：応用

汎用UARTドライバが受信完了時に製品固有関数を直接呼び出しており、他製品へ再利用しにくい。改善として最も適切なものはどれか。

ア．上位層がコールバック関数や通知先を登録できる汎用インタフェースにし、製品固有処理をドライバから分離する。

イ．ドライバへ全製品のアプリ処理を追加する。

ウ．受信完了通知を廃止する。

エ．ドライバ内からグローバル関数名を固定参照する。

答え・解説 正答：ア

ア：ドライバが上位アプリを直接知る依存をなくすと再利用しやすい。
イ：製品依存が増える。
ウ：非同期通信を扱いにくくなる。
エ：再利用先で結合度が高くなる。
総合解説：ドライバが上位アプリを直接知る依存をなくすと再利用しやすい。 この問題では「Callback設計」として、ドライバ再利用でコールバックとコンテキストを用いて上位依存を減らせる。

対象：2026年度 / 基準日 2026-09-02

0120 3-4 ソフトウェア設計・開発手法 > インタフェース・タスク間通信・再利用 ｜ 難易度：応用

制御タスクからログタスクへ少量イベントを送る場合と、画像処理タスクへ1MBフレームを渡す場合で同じ通信方式を使おうとしている。適切な設計判断はどれか。

ア．全てのデータを同じ1バイトQueueへ入れる。

イ．少量イベントはQueue/Notification等、巨大データはPool＋ハンドル渡し等、データ特性に応じて通信方式を選ぶ。

ウ．大容量データを毎回何重にもコピーする。

エ．イベントは共有グローバル変数へ無保護で上書きする。

答え・解説 正答：イ

ア：大容量データを扱えない。
イ：コピーコスト、待機、所有権、複数受信者などの特性に応じてIPC方式を使い分けるべきである。
ウ：CPU時間とメモリ帯域を浪費する。
エ：取りこぼしや競合が起こり得る。
総合解説：コピーコスト、待機、所有権、複数受信者などの特性に応じてIPC方式を使い分けるべきである。 この問題では「IPC方式選定」として、通信方式選択でデータ量・遅延・所有権・多対多性を比較できる。

対象：2026年度 / 基準日 2026-09-02

0121 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：基礎

単体テストの主な対象として最も適切なものはどれか。

ア．最終製品を市場環境だけで確認すること。
イ．複数ECUを実車へ搭載した総合試験だけ。
ウ．個々の関数、クラス、モジュールなど比較的小さなソフトウェア単位。
エ．ハードウェア製造ラインの外観検査だけ。

答え・解説 正答：ウ

ア：これはシステム・受入レベルに近い。
イ：単体より上位の統合・システム試験である。
ウ：単体テストでは個別コンポーネントのロジックを他部分から可能な限り分離して確認する。
エ：ソフトウェア単体テストではない。
総合解説：単体テストでは個別コンポーネントのロジックを他部分から可能な限り分離して確認する。 この問題では「単体テスト」として、単体テストの対象が個々のコンポーネントやモジュールであることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0122 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：基礎

静的テストの説明として最も適切なものはどれか。

ア．必ず実機上で全コードを実行する。
イ．テスト対象を一切確認しない。
ウ．静的テストはソースコードだけに限定される。
エ．ソースコード、要求、設計書などを実行せずレビューや静的解析で欠陥を検出する。

答え・解説 正答：エ

ア：これは動的テストである。
イ：欠陥検出活動にならない。
ウ：要求仕様や設計文書等も対象になり得る。
エ：ISTQBでは静的テストはソフトウェアを実行せず成果物を評価する活動として扱われる。
総合解説：ISTQBでは静的テストはソフトウェアを実行せず成果物を評価する活動として扱われる。 この問題では「静的テスト」として、静的テストがプログラムを実行せず成果物を評価することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0123 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：基礎

入力値1〜100が有効で、それ以外が無効な関数をテストする。同値分割の考え方として適切なものはどれか。

ア．1〜100の有効クラスと、0以下・101以上などの無効クラスから代表値を選ぶ。
イ．1から100まで全ての値だけを必ず試すことが同値分割である。
ウ．境界値だけを一つ選ぶことを意味する。
エ．ランダム値だけを根拠なく選ぶ。

答え・解説 正答：ア

ア：同じ振る舞いを期待する入力集合を同値クラスとしてまとめ、代表値で効率よくテストする。
イ：全数試験ではなくクラス代表値を選ぶ技法である。
ウ：境界値分析とは別技法である。
エ：同値クラスに基づく設計ではない。
総合解説：同じ振る舞いを期待する入力集合を同値クラスとしてまとめ、代表値で効率よくテストする。 この問題では「同値分割」として、同値分割で代表値を選びテストケース数を抑えられることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0124 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：標準

入力範囲が0〜255の8ビット値として仕様化されている。境界値分析で優先して確認すべき組合せはどれか。

ア．100だけ。
イ．0、1、254、255に加え、入力型や外部I/Fで表現可能なら-1や256相当の異常条件。
ウ．50と150だけ。
エ．常に0だけ。

答え・解説 正答：イ

ア：境界条件を十分確認できない。
イ：境界付近は不等号やオーバフローの欠陥が生じやすいため重点確認する。
ウ：内部代表値であり境界分析にならない。
エ：上限側の欠陥を検出できない。
総合解説：境界付近は不等号やオーバフローの欠陥が生じやすいため重点確認する。 この問題では「境界値分析」として、境界値分析で有効範囲の境界付近を重点テストできる。

対象：2026年度 / 基準日 2026-09-02

0125 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：標準

制御関数の単体テストで、実I2Cセンサから特定エラーを再現しにくい。適切な方策はどれか。

ア．実センサで偶然エラーが起きるまで待つ。
イ．異常系テストを削除する。
ウ．センサドライバをStub/Mockへ置換し、正常値・タイムアウト・CRCエラー等を任意に返す。
エ．制御関数へセンサレジスタ操作を追加する。

答え・解説 正答：ウ

ア：再現性が低く非効率である。
イ：重要な故障処理を検証できない。
ウ：依存先をテストダブルへ差し替えると異常系を再現可能かつ自動化しやすい。
エ：依存がさらに強くなる。
総合解説：依存先をテストダブルへ差し替えると異常系を再現可能かつ自動化しやすい。 この問題では「Test Double」として、スタブとモックの利用で依存先を置換して単体テストできる。

対象：2026年度 / 基準日 2026-09-02

0126 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：標準

静的解析ツールが100件の警告を出した。適切な運用はどれか。

ア．警告が多いので全て無視する。
イ．警告が1件でもあれば製品は必ず動作不能と判断する。
ウ．抑制理由を記録しない。
エ．警告の種類・重大度・到達可能性を確認し、真の欠陥を修正し、抑制する場合は根拠を記録する。

答え・解説 正答：エ

ア：重大な欠陥を見逃す。
イ：警告の意味と実現性を評価する必要がある。
ウ：後の監査・変更時に判断根拠を追えない。
エ：静的解析は自動検出に有効だが誤検知もあり得るため、人によるトリアージとルール管理が必要である。
総合解説：静的解析は自動検出に有効だが誤検知もあり得るため、人によるトリアージとルール管理が必要である。 この問題では「静的解析警告」として、静的解析警告を重大度・実現可能性に基づきレビューできる。

対象：2026年度 / 基準日 2026-09-02

0127 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：標準

ステートメントカバレッジ100%を達成した単体テストについて最も適切な説明はどれか。

ア．全ステートメントを少なくとも一度実行したことを示すが、全条件・境界・要求が十分に検証された保証にはならない。

イ．欠陥が0件であることを保証する。

ウ．全入力組合せを試したことを意味する。

エ．性能要求を満たしたことを保証する。

答え・解説 正答：ア

ア：カバレッジは網羅性の指標の一つであり、要求ベーステストや条件組合せなど他観点も必要である。

イ：100%カバレッジでも欠陥は残り得る。

ウ：ステートメントカバレッジは入力全数試験ではない。

エ：機能実行範囲の指標であり性能保証ではない。

総合解説：カバレッジは網羅性の指標の一つであり、要求ベーステストや条件組合せなど他観点も必要である。 この問題では「Code Coverage」として、コードカバレッジがテスト品質の一側面であり十分性を単独保証しないことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0128 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：標準

通信ドライバのタイムアウト処理を修正した。変更箇所の確認に加えて必要なテストとして最も適切なものはどれか。

ア．修正した1行だけを目視しテストしない。

イ．既存の正常通信、他エラー処理、上位モジュールとの連携に影響がないか回帰テストを行う。

ウ．全ての既存テストを削除する。

エ．出荷後の利用者報告だけで確認する。

答え・解説 正答：イ

ア：副作用を検出できない。

イ：変更は周辺機能へ副作用を与える可能性があるため、既存動作の維持も確認する。

ウ：回帰確認の基盤を失う。

エ：開発段階で検証すべきである。

総合解説：変更は周辺機能へ副作用を与える可能性があるため、既存動作の維持も確認する。 この問題では「Regression Test」として、回帰テストで変更による既存機能への副作用を確認できる。

対象：2026年度 / 基準日 2026-09-02

0129 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：応用

符号付き整数のオーバフローが疑われる演算を含む安全関連Cコードを検証する。適切な方策はどれか。

ア．通常値だけを1回テストする。

イ．警告を全て抑制し理由を残さない。

ウ．静的解析で危険演算を検出し、境界値テストと必要な範囲チェックを追加して実行時挙動も確認する。

エ．オーバフローは組込みCでは起こらないと仮定する。

答え・解説 正答：ウ

ア：境界条件を確認できない。

イ：欠陥を隠す可能性がある。

ウ：静的解析と動的境界テストを組み合わせると、実行経路と潜在的危険の両面から検証できる。

エ：有限ビット幅のため発生し得る。

総合解説：静的解析と動的境界テストを組み合わせると、実行経路と潜在的危険の両面から検証できる。 この問題では「C言語境界欠陥」として、未定義動作や境界条件を静的解析と単体テストの両面で検出できる。

対象：2026年度 / 基準日 2026-09-02

0130 3-5 テスト・統合・構成管理 > 単体テスト・静的解析 ｜ 難易度：応用

単体テスト完了判定を、単にテスト件数だけで決めようとしている。より適切な判断はどれか。

ア．テストケースが100件あれば内容に関係なく合格とする。

イ．正常系が1件成功すれば完了とする。

ウ．未解決重大欠陥があってもカバレッジ100%なら完了とする。

エ．要求カバレッジ、コードカバレッジ、境界・異常系、未解決欠陥、静的解析結果、変更回帰状況を総合して出口基準を判定する。

答え・解説 正答：エ

ア：件数だけでは網羅性を示せない。

イ：異常系や境界条件を確認できない。

ウ：重大欠陥の残存を無視できない。

エ：単一指標だけでは品質を十分説明できないため、複数の客観的基準を組み合わせる。

総合解説：単一指標だけでは品質を十分説明できないため、複数の客観的基準を組み合わせる。 この問題では「単体テスト完了判定」として、要求・テスト・カバレッジ・欠陥を組み合わせて単体品質を総合評価できる。

対象：2026年度 / 基準日 2026-09-02

0131 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：基礎

結合テストの主な目的として最も適切なものはどれか。

ア．複数のコンポーネントを組み合わせ、インタフェースやデータ受渡し、相互作用の欠陥を検出する。

イ．関数一つだけの内部ロジックを確認することだけ。

ウ．市場投入後の利用状況だけを観察する。

エ．部品の外観だけを検査する。

答え・解説 正答：ア

ア：個々の単体が正しくても接続条件や順序、データ形式の不整合で欠陥が起こり得る。

イ：単体テストの説明である。

ウ：開発段階の結合検証ではない。

エ：ソフトウェア結合試験ではない。

総合解説：個々の単体が正しくても接続条件や順序、データ形式の不整合で欠陥が起こり得る。 この問題では「Integration Test」として、結合テストがモジュール間インタフェースと相互作用を主対象にすることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0132 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：基礎

HIL（Hardware-in-the-Loop）テストの説明として最も適切なものはどれか。

ア．制御モデルだけをPC上で動かす。

イ．実際のECUをリアルタイムシミュレータへ接続し、車両やセンサ・アクチュエータ等の環境を模擬して検証する。

ウ．ソースコードを静的解析するだけ。

エ．量産基板を外観検査するだけ。

答え・解説 正答：イ

ア：MILに近い。

イ：HILではテスト対象ハードウェアを現実的なシミュレーション環境へ組み込み、再現性のある検証を行う。

ウ：HILは動的な実機連携試験である。

エ：HILの説明ではない。

総合解説：HILではテスト対象ハードウェアを現実的なシミュレーション環境へ組み込み、再現性のある検証を行う。 この問題では「HIL基本」として、HILが実ECUをリアルタイムシミュレータ環境へ接続して検証する手法であることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0133 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：基礎

HILテストの代表的な利点として正しいものはどれか。

ア．実際のECUハードウェアを一切使えない。
イ．シミュレーション結果は毎回必ず異なる必要がある。
ウ．実車・実機で危険な故障条件やコーナーケースを、ラボで再現性高く自動試験できる。
エ．故障注入は実施できない。

答え・解説 正答：ウ

ア：HILでは実ECUを使用する。
イ：再現性はむしろ利点である。
ウ：dSPACEはHILの利点として再現性、自動化、安全に重大なコーナーケースを扱える点を挙げている。
エ：HILでは電気的故障等を模擬できる。
総合解説：dSPACEはHILの利点として再現性、自動化、安全に重大なコーナーケースを扱える点を挙げている。 この問題では「HIL利点」として、HILで危険なコーナーケースを安全・再現可能に試験できる利点を理解できる。

対象：2026年度 / 基準日 2026-09-02

0134 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：標準

上位制御ロジックから結合を開始したいが、下位センサドライバが未完成である。適切なテスト方策はどれか。

ア．未完成でも不定値を返す実装をそのまま使う。
イ．上位テストを全て中止し、全ソフト完成まで待つ。
ウ．Stubはテストで使用できない。
エ．下位ドライバをStubで代替し、上位モジュールの結合を先行して確認する。

答え・解説 正答：エ

ア：テスト結果の再現性を失う。
イ：段階的統合の利点を失う。
ウ：代表的なテストダブルである。
エ：トップダウン結合では未完成下位モジュールをStubで置き換えて統合を進められる。
総合解説：トップダウン結合では未完成下位モジュールをStubで置き換えて統合を進められる。 この問題では「結合戦略」として、トップダウン結合で未完成下位モジュールをStubで代替できる。

対象：2026年度 / 基準日 2026-09-02

0135 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：標準

CANネットワーク上のECUをHILで試験するが、他ECUの多くはまだ利用できない。適切な方策はどれか。

ア．HILシミュレータで未接続ECUのバス通信をレストバスシミュレーションする。
イ．通信相手がないまま全受信要求を成功扱いする。
ウ．CANを全てUARTへ置換する。
エ．他ECUが完成するまで通信テストを一切行わない。

答え・解説 正答：ア

ア：レストバスシミュレーションにより、存在しないネットワークノードのメッセージや応答を模擬できる。
イ：通信条件を検証できない。
ウ：試験対象インタフェースが変わってしまう。
エ：フロントローディングの利点を失う。
総合解説：レストバスシミュレーションにより、存在しないネットワークノードのメッセージや応答を模擬できる。 この問題では「Restbus Simulation」として、レストバスシミュレーションで未接続ECUの通信を模擬できる。

対象：2026年度 / 基準日 2026-09-02

0136 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：標準

HILでセンサ入力線を短絡・断線させ、ECUの診断と安全動作を確認する試験の目的として最も適切なものはどれか。

ア．ECUの通常機能を無条件に無効化することだけ。
イ．故障時に要求どおり異常を検出し、DTC記録や安全状態移行が行われるか検証する。
ウ．実際の故障を永久に残す。
エ．ソースコード行数を測る。

答え・解説 正答：イ

ア：故障時要求を検証することが目的である。
イ：故障注入は正常条件では再現しにくい異常系の診断・フェールセーフ動作確認に有効である。
ウ：HILでは制御された模擬故障を再現可能に与える。
エ：故障時動作の検証とは無関係である。
総合解説：故障注入は正常条件では再現しにくい異常系の診断・フェールセーフ動作確認に有効である。 この問題では「Fault Injection」として、故障注入でセンサ断線や短絡時の診断・安全動作を検証できる。

対象：2026年度 / 基準日 2026-09-02

0137 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：標準

組込みシステムのシステム検証項目として最も適切なものはどれか。

ア．正常系の一つの画面表示だけを確認する。

イ．単体テストが通ればシステム検証は不要とする。

ウ．機能、通信、性能・タイミング、起動停止、異常系、資源使用量などシステム要求に対応する項目を確認する。

エ．要求との対応を管理しない。

答え・解説 正答：ウ

ア：システム要求全体をカバーできない。
イ：結合・実環境依存の欠陥が残り得る。
ウ：システムレベルでは統合された製品が要求仕様を満たすかを総合的に検証する。
エ：検証漏れを確認できない。
総合解説：システムレベルでは統合された製品が要求仕様を満たすかを総合的に検証する。 この問題では「System Verification」として、システムテストで機能要件だけでなく性能・タイミング・障害処理を確認できる。

対象：2026年度 / 基準日 2026-09-02

0138 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：標準

HILではECUが正常だが実車では制御が不安定になる。最も疑うべき事項の一つはどれか。

ア．HILを使えばモデル誤差は原理上存在しない。

イ．実車結果は常に誤りと判断する。

ウ．HILではタイミングを考慮する必要がない。

エ．HIL内のプラントモデル、センサ遅延、I/Oスケーリング、リアルタイム刻みが実環境を十分再現しているか。

答え・解説 正答：エ

ア：モデル化誤差は残り得る。
イ：双方の差異原因を分析する必要がある。
ウ：リアルタイム性は重要な要素である。
エ：HIL結果の信頼性はシミュレータ側モデルとI/Oの妥当性にも依存する。
総合解説：HIL結果の信頼性はシミュレータ側モデルとI/Oの妥当性にも依存する。 この問題では「HILモデル妥当性」として、HILモデルの妥当性が試験結果の信頼性へ影響することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0139 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：応用

100個のモジュールを一度に全結合した後、大量の障害が発生して原因特定が困難になった。改善方針として最も適切なものはどれか。

ア．依存関係とリスクに基づき小さな単位で段階的に統合し、各段階で結合テストを実施する。

イ．さらに多くの変更を同時投入する。

ウ．結合テストを削除する。

エ．障害を記録せず再起動だけ行う。

答え・解説 正答：ア

ア：段階的統合なら直前に追加した要素へ原因候補を絞りやすく、早期にインタフェース欠陥を発見できる。

イ：原因切分けがさらに困難になる。

ウ：統合欠陥を検出できない。

エ：原因分析ができない。

総合解説：段階的統合なら直前に追加した要素へ原因候補を絞りやすく、早期にインタフェース欠陥を発見できる。 この問題では「Incremental Integration」として、段階的統合で不具合の切分け容易性を高められる。

対象：2026年度 / 基準日 2026-09-02

0140 3-5 テスト・統合・構成管理 > 結合・HIL・システム検証 ｜ 難易度：応用

安全要求「センサ断線から100ms以内に出力を停止」をHILで検証する。適切な試験設計はどれか。

ア．断線を注入せず正常運転だけ確認する。

イ．断線故障を制御時刻で注入し、診断検出時刻と出力停止時刻を計測し、要求100ms以内かを自動判定して要求IDへ結果を紐付ける。

ウ．停止までの時間を測らず目視だけで判定する。

エ．試験結果を要求と紐付けない。

答え・解説 正答：イ

ア：対象安全要求を検証していない。

イ：要求条件、刺激、測定量、判定基準を対応付けることで定量的かつ追跡可能な検証になる。

ウ：100ms要求を定量確認できない。

エ：検証網羅性と変更影響を追跡しにくい。

総合解説：要求条件、刺激、測定量、判定基準を対応付けることで定量的かつ追跡可能な検証になる。 この問題では「HIL要求トレーサビリティ」として、要求からHILシナリオまでトレーサビリティを持たせて検証網羅性を評価できる。

対象：2026年度 / 基準日 2026-09-02

0141 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：基礎

ソフトウェア構成管理でバージョン管理システムを用いる主な目的はどれか。

ア．最新ファイルだけ残し過去履歴を全て削除する。

イ．変更者情報を記録しない。

ウ．ソースコードや設定の変更履歴、作成者、差分、分岐・統合を追跡し、特定版を再現できるようにする。

エ．全開発者が同じ作業コピーを上書きする。

答え・解説 正答：ウ

ア：変更追跡と再現性を失う。

イ：監査性が低下する。

ウ：構成管理では変更を識別・制御し、必要な版を再現可能にすることが重要である。

エ：競合や履歴喪失の原因になる。

総合解説：構成管理では変更を識別・制御し、必要な版を再現可能にすることが重要である。 この問題では「Version Control」として、バージョン管理がソースや設定の変更履歴を追跡する基盤であることを理解できる。

対象：2026年度 / 基準日 2026-09-02

0142 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：基礎

Gitのブランチを利用する代表的な目的として最も適切なものはどれか。

ア．ブランチを作成すると全ファイルが必ず物理的に複製される。

イ．ブランチを作成すると履歴が消える。

ウ．ブランチではマージできない。

エ．主系統へ影響を与えず、機能開発や修正を独立した履歴系列で進め、後で統合する。

答え・解説 正答：エ

ア：Gitブランチは軽量な参照である。

イ：履歴を保持したまま分岐する。

ウ：分岐した履歴をマージできる。

エ：Gitのブランチは軽量なポインタとして変更系列を分離し、並行開発を支援する。

総合解説：Gitのブランチは軽量なポインタとして変更系列を分離し、並行開発を支援する。 この問題では「Git Branch」として、Gitブランチが変更系列を分離して並行開発を支援することを理解できる。

対象：2026年度 / 基準日 2026-09-02

0143 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：基礎

構成管理におけるベースラインの説明として最も適切なものはどれか。

ア．レビュー・承認された構成項目の特定時点の版を基準として固定し、以後の変更を管理対象とする。

イ．開発中の任意ファイルを無管理で上書きすること。

ウ．全履歴を削除した状態。

エ．テスト結果だけを保存しソース版は記録しない。

答え・解説 正答：ア

ア：ベースラインは再現可能な基準版を定義し、変更前後の差分を明確にする。
イ：ベースライン管理とは逆である。
ウ：再現性を失う。
エ：成果物間の対応を再現できない。
総合解説：ベースラインは再現可能な基準版を定義し、変更前後の差分を明確にする。 この問題では「Baseline」として、リリースベースラインが承認済み構成を固定する役割を理解できる。

対象：2026年度 / 基準日 2026-09-02

0144 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：標準

量産直前に通信仕様変更要求が提出された。最も適切な変更管理手順はどれか。

ア．担当者が口頭で了承したら即座に本番コードへ直接修正する。

イ．要求理由、影響範囲、リスク、工数、必要テストを分析し、承認後に変更・検証し、関連構成項目と文書を更新する。

ウ．変更後にテストを行わない。

エ．関連仕様書を更新しない。

答え・解説 正答：イ

ア：影響分析や監査証跡がなくリスクが高い。
イ：変更を統制された手順で扱うことで無承認変更や検証漏れを防ぐ。
ウ：副作用や要求適合を確認できない。
エ：構成間の不整合が残る。
総合解説：変更を統制された手順で扱うことで無承認変更や検証漏れを防ぐ。 この問題では「Change Control」として、変更要求を影響分析・承認・実施・検証まで管理する必要性を判断できる。

対象：2026年度 / 基準日 2026-09-02

0145 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：標準

1ms周期タスクの性能測定結果が平均600μs、最大1.2msだった。デッドラインが1msの場合の判断はどれか。

ア．平均600μsなので必ず合格である。

イ．最大値はタイミング検証では使用しない。

ウ．平均は余裕があるが最大値で期限違反しているため、最悪条件の原因分析と改善が必要である。

エ．周期1msなら実行時間は測定不要である。

答え・解説 正答：ウ

ア：最大1.2msの期限違反を無視している。

イ：最悪応答の重要指標である。

ウ：リアルタイム要件は平均ではなく各ジョブの期限保証が重要である。

エ：期限保証のため測定・解析が必要である。

総合解説：リアルタイム要件は平均ではなく各ジョブの期限保証が重要である。 この問題では「Timing Verification」として、タイミング検証で平均値だけでなく最大値・ジッタ・期限違反を確認できる。

対象：2026年度 / 基準日 2026-09-02

0146 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：標準

半年前の量産ファームウェアを同一バイナリとして再ビルドしたい。構成管理で保存すべき情報として最も適切なものはどれか。

ア．ソースファイル名だけ。

イ．担当者の記憶だけ。

ウ．最新ツールへ自動置換したことだけ。

エ．ソース版、依存ライブラリ、コンパイラ/リンカ版、ビルド設定、生成スクリプト、設定データなど。

答え・解説 正答：エ

ア：同じソースでもツール版や設定で生成物が変わり得る。

イ：再現性・監査性がない。

ウ：過去版再現には当時の構成を特定する必要がある。

エ：再現可能なビルドにはソースだけでなくツールチェーンと依存物の版・設定も必要になる。

総合解説：再現可能なビルドにはソースだけでなくツールチェーンと依存物の版・設定も必要になる。 この問題では「Build Reproducibility」として、リリース再現にソースだけでなくツールチェーン・設定・生成物版が必要なことを理解できる。

対象：2026年度 / 基準日 2026-09-02

0147 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：標準

ログ機能追加後、機能テストは合格したがリアルタイム制御への影響が懸念される。追加で行うべき確認はどれか。

ア．CPU使用率、WCET、割込みレイテンシ、タスク応答時間、ジッタなどのタイミング回帰テスト。

イ．ログ内容だけ確認して完了する。

ウ．既存タイミング測定値をそのまま使う。

エ．制御タスクの期限を削除する。

答え・解説 正答：ア

ア：機能追加はCPU・I/O負荷や排他時間を変え、既存期限へ影響する可能性がある。

イ：非機能的な時間影響を確認できない。

ウ：変更後の実測・解析が必要である。

エ：要求変更なしには不適切である。

総合解説：機能追加はCPU・I/O負荷や排他時間を変え、既存期限へ影響する可能性がある。 この問題では「Timing Regression」として、変更後に対象機能だけでなくタイミング回帰を実施する必要性を判断できる。

対象：2026年度 / 基準日 2026-09-02

0148 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：標準

一つのコミットに機能追加、リファクタリング、フォーマット変更、ビルド設定変更を大量に混在させてレビューが困難になった。改善として適切なものはどれか。

ア．さらに無関係な変更を同じコミットへ追加する。

イ．論理的に独立した変更単位へコミットを分け、目的が分かるメッセージと関連課題を紐付ける。

ウ．コミットメッセージを全て空欄にする。

エ．レビュー前に履歴を削除する。

答え・解説 正答：イ

ア：レビュー困難を悪化させる。

イ：変更単位を小さくすると差分レビュー、原因切分け、必要時の切戻しが容易になる。

ウ：履歴の理解が難しくなる。

エ：監査性を失う。

総合解説：変更単位を小さくすると差分レビュー、原因切分け、必要時の切戻しが容易になる。 この問題では「Atomic Change」として、変更セットを小さく保ちレビュー・切戻し容易性を高められる。

対象：2026年度 / 基準日 2026-09-02

0149 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：応用

市場不具合のあった製品からファームウェア版番号だけは取得できたが、その版で実施したテスト結果や設定ファイルが特定できない。改善策として最も適切なものはどれか。

ア．版番号だけ付け、他情報は管理しない。

イ．テスト結果を上書きして常に最新だけ残す。

ウ．リリースIDからソースコミット、依存物、設定、ビルド生成物、テスト結果、承認記録を一意に追跡できるよう構成管理する。

エ．市場不具合時にソースを推測して再作成する。

答え・解説 正答：ウ

ア：再現に必要な構成を特定できない。
イ：過去版の証跡が失われる。
ウ：製品版と検証証跡を対応付けると、障害再現と影響分析を迅速に行える。
エ：正確な再現ができない。
総合解説：製品版と検証証跡を対応付けると、障害再現と影響分析を迅速に行える。 この問題では「Release Traceability」として、リリース構成とテスト証跡を対応付けて再現性を確保できる。

対象：2026年度 / 基準日 2026-09-02

0150 3-5 テスト・統合・構成管理 > タイミング検証・構成管理・変更管理 ｜ 難易度：応用

RTOS設定でTick周波数とタスク優先度を変更した。機能テストは全て合格している。リリース前の判断として最も適切なものはどれか。

ア．機能テストが通ったのでタイミング確認は不要とする。

イ．変更した設定値を記録しない。

ウ．以前のタイミング結果をそのまま承認資料へ使う。

エ．変更影響として周期精度、CPU負荷、コンテキストスイッチ、タイムアウト単位、最悪応答時間を再評価し、構成版と結果をベースライン化する。

答え・解説 正答：エ

ア：期限違反や負荷増加を見逃す可能性がある。
イ：再現性が失われる。
ウ：変更後の条件を反映していない。
エ：RTOS設定変更は機能結果が同じでもタイミング特性を変えるため、非機能回帰と構成管理が必要である。
総合解説：RTOS設定変更は機能結果が同じでもタイミング特性を変えるため、非機能回帰と構成管理が必要である。 この問題では「Release Change Assessment」として、変更管理とリアルタイム検証を統合して安全なリリース判定ができる。

対象：2026年度 / 基準日 2026-09-02