# ECE 460/560 Fall 2025
# Theory Homework 2 Solutions v3

This homework considers code and peripherals for the NXP KL25Z MCU with an Arm Cortex-M0+ CPU core. Some questions below include references to the **KL25Z Subfamily Reference Manual** (https://wordpress-courses2527.wolfware.ncsu.edu/ece-460560-fall-2025-emb-sys-arch/wp-content/uploads/sites/27/2025/09/KL25P80M48SF0RM-Rev-3.pdf) and **Embedded Systems Fundamentals** for further details needed to complete this assignment.

## Race Conditions

```
volatile int32_t pos = 0; // in memory

void ISR_ZL(void) {
    pos = 0;
    // IZ.1 Zero out r1
    // IZ.2 Store r1 to pos
}

void ISR_QD(void) {
    read Z
    if (Z == CLOSED) {
       pos = 0;
       // IQ.1 Zero out r0
       // IQ.2 Store r0 to pos
    } else {
       read B
       if (B == 0)
          pos = pos+1;
          // IQ.10 Load r0 from pos
          // IQ.11 Add 1 to r0
          // IQ.12 Store r0 to pos
       else
          pos = pos-1;
          // IQ.20 Load r0 from pos
          // IQ.21 Add 1 to r0
          // IQ.22 Store r0 to pos
    }
}
```

```
void T1 ( void ) {
    int32_t target, pos_error, motor_dir;
    ...
    while (1) {
       target = Read_UI();
       pos_error = target - pos;
       // T1.10 Load r2 from target
       // T1.11 Load r3 from pos
       // T1.12 Subtract: r4 = r3-r2
       if (pos_error > 10)
          motor_dir = 1; // forward
       else if (pos_error < -10)
          motor_dir = -1; // reverse
       else
          motor_dir = 0; // stop
       Control_Motor(motor_dir);
    }
}
```

This code builds on the quadrature decoder by allowing calibration to a fixed position using a limit switch and code. When the limit switch is open, the quadrature decoder interprets encoder signals A and B to increment or decrement the current position estimate. When the limit switch is closed, the position is fixed at zero and the quadrature decoder ignores signals A and B. The code has one thread (T1) and two interrupt service routines. ISR_ZL (short name IZ) runs after each time the zero limit switch closes (goes from OPEN t1o CLOSED). ISR_QD (short name IQ) runs after each positive transition on signal A.

Arm ISA assembly pseudo-code instructions are listed after certain lines of source code. For more details on the Arm ISA, please refer to Chapter 4 of **ESF**. Assume that the ISRs and T1 use different registers in the CPU core (r0, r1, etc.). This code has at least one data race bug which will result in the wrong value being stored in variable **pos** after execution. The bug can be triggered by certain orderings and interleavings of instructions.

1. How many critical sections for **pos** are in:
   a. ISR_ZL
      1 critical section (IZ.2 writes **pos).**
   b. ISR_QD
      3 critical sections: (IQ.2), (IQ.10, IQ.11, IQ.12) and (IQ.20, IQ.21, IQ.22): each section reads and then possibly writes **pos)**.
   c. T1
      This depends on whether it matters if pos changes **after the load (T1.11) but before it is used (T.12).** Either is accepted for full credit.

      **Yes? 1 critical section.**
      - T1.11, T1.12
      - T1.11 also ok

      **No? 0 critical sections.**
      - Instruction T1.11 in Thread T1 reads the memory location of the variable **pos** to load register r3. T1 does not write anything to that memory location. So, only one instruction (T1.11) **might** be in a critical section for **pos**.
      - If T1.11 were to execute between IQ.10 and IQ.12 (or between IQ.20 and IQ.22), then T1 would be *reading* **pos** during an update by ISR_QD. This would lead to T1 getting the previous value of pos, which would make T1.10 a critical section.
      - However, T1 is a thread, and ISR_QD is an interrupt service routine. Threads are lower priority than interrupt service routines, so T1 cannot preempt ISR_QD, so T1.11 will never execute between IQ.10 and IQ.12, or IQ.20 and IQ.22.

      Which of these is correct is unclear, but **Yes** is probably correct.
      It may be defined by the programming language semantics, or it may be left undefined (compiler-dependent). It is a complex topic; here are some discussions:

      - *Common Compiler Optimisations are Invalid in the C11 Memory Model and what we can do about it*, https://fzn.fr/readings/c11comp.pdf
      - *The Problem of Programming Language Concurrency Semantics*, https://www.cl.cam.ac.uk/~jp622/the_problem.pdf
      - *Programming language memory models*: https://research.swtch.com/plmm
      - *Hardware memory models*: https://research.swtch.com/hwmm

2. List the instructions in three critical sections for variable **pos** in each of these ISRs or threads. Be sure not to include extra instructions in the critical sections. Use the instruction identifiers (e.g. IZ.1):

   IZ.2
   IQ.2
   IQ.10, IQ.11, IQ.12
   IQ.20, IQ.21, IQ.22
   T1.10 T1.11 T1.12
   T1.10 T2.11 T2.12
   Many students entered all three critical sections in first response box.

3. Give one example of an instruction sequence which triggers this data race bug. Both ISRs can preempt thread T1, and each ISR can interrupt the other. List the instruction sequence (in order) which will trigger the bug. Fill in the table below with the instruction identifiers. The columns marked **-** are for your notes and scratch work (if needed). Only entries in T1, ISR_QD and ISR_ZL will be graded.

| Exec. Order | Variable in Memory | T1 | - | ISR_QD (IQ.#) | - | ISR_ZL (IZ.#) | - |
|---|---|---|---|---|---|---|---|
| 1 | 6 | | | IQ.10 | r0=6 | | |
| 2 | 6 | | | | | IZ.1 | r1=0 |
| 3 | 6, then 0 | | | | | IZ.2 | |
| 4 | 0 | | | IQ.11 | r0=7 | | |
| 5 | 0, then 7 | | | IQ.12 | | | |

Ordering for key instructions in race sequences (omits intervening instructions, e.g. IQ.11 in 1st bullet)

- IQ.10 IZ.2 IQ.12
- IQ.20 IZ.2 IQ.22
- T1.11 IZ.2 T1.12
- T1.11 IQ.12 T1.12
- T1.11 IQ.22 T1.12

4. **ECE 560 Only:** List three other different sequences of instruction execution which trigger the data race bug.

   The text section in Google Form was incorrectly created as question, so 5 free points.

   Example sequences, but many more are possible (including those based on #3 above).

   a. Sequence 2
      IQ.10, IQ.11, IZ.1, IZ.2, IQ.12
   b. Sequence 3
      IQ.20, IZ.1, IZ.2, IQ.21, IQ.22
   c. Sequence 4
      IQ.20, IQ.21, IZ.1, IZ.2, IQ.22
      ***Full credit for ISR changing value of pos after T1 reads but before T1 uses it***

# Peripherals

## Event Detection

Consider the UART1 peripheral, described in Chapter 40 of the **SRM**. It is able detect certain events and then generate service requests (interrupt or Direct Memory Access). Start by reading Sections 40.1.3, 40.3.4 and 40.3.5 to answer these questions.

The UART1 peripheral has ten status flags as possible interrupt sources. Three of these sources can be used to build a more complex protocol on top of what the UART provides: **IDLE, RXEDGIF** and **LBKDIF**. The other seven status flag interrupt sources are **TDRE, TC, RDRF, OR, FE, NF** and **PF**. Which of these sources…

5. Synchronize transfer of data to the UART for serial **transmission**?
   TDRE, TC
6. Synchronize transfer of data from the UART for serial **reception**?
   RDRF
7. Indicate error conditions?
   OR, FE, NF, PF

## Hardware and the Processing Chain

The MCU's Analog to Digital Converter (ADC) is described in Chapter 28 of the SRM. You will evaluate hardware features in the event processing chain. Start by reading Section 28.1.1 in the SRM for a list of the features and a block diagram. Then skim Section 28.4 (ADC functional description) and its subsections for an overview of the behavior and use of the ADC and its features.

Each feature may operate in one or more steps in the processing chain. For each row's feature, mark its step(s). Note that the ADC must be triggered in order to start a conversion. A column for this synchronization stage (ADC Triggering) has been added before the event detection stages.

| Analog to Digital Converter Feature | Triggering | Data Conversion | | Event Detection from Converted Data | | Schedule & Dispatch | Processing Work / Handler |
|---|---|---|---|---|---|---|---|
| | Start ADC Conversion | Sample | Quantize | Analyze | Decide | | |
| 8. Linear successive approximation algorithm with up to 16-bit resolution | | | Y | | | | |
| 9. Configurable sample time and conversion speed/power | | Y | (ok) | | | | |
| 10. Output modes: Differential 16-bit, 13-bit, 11-bit, and 9-bit modes, single-ended 16-bit, 12-bit, 10-bit, and 8-bit modes | | | Y | | | | |
| 11. Single or continuous conversion, that is, automatic return to idle after single conversion | Y | | | | | | |
| 12. Conversion complete flag and interrupt | | | | Y | Y | Y | (ok) |
| 13. Software conversion trigger option | Y | | | | | | |
| 14. Selectable hardware conversion trigger with hardware channel select | Y | (ok) | | | | | |
| 15. Automatic compare with interrupt for less-than, greater-than or equal-to, within range, or out-of-range, programmable value | | | | Y | Y | Y | (ok) |
| 16. Hardware average function | | | | Y | | | (ok) |
| 17. **ECE 560 Only:** Compensation for errors in offset, plus-side gain and minus-side gain | | | Y | Y | | | (ok) |
| 18. **ECE 560 Only:** Self-Calibration mode to determine correction values for offset, plus-side gain and minus-side gain | Y | Y | Y | Y | Y | | (ok) |
| 19. **ECE 560 Only:** Selectable voltage reference: external or alternate | | | Y | | | | |

# Grading

| | Question | Q Graded In... Google Forms | Spread sheet | Pts | Comments |
|---|---|---|---|---|---|
| | # Critical Sections | 1 | | 15 | |
| | Instr. IDs | | 2 | 15 | |
| | Data race example | | 3 | 5 | Partial credit for preemption of T1, nonexistent instructions ... |
| Other Data Races | ODR intro (free points) | 4 | | 5 | |
| | Other data races | | 4a | 5 | Full credit given for reasonable responses |
| | Other data races | | 4b | 5 | Full credit given for reasonable responses |
| | Other data races | | 4c | 5 | Full credit given for reasonable responses |
| Sync Intrpt | Sync for data tx | 5 | | 5 | |
| | Sync for data rx | 6 | | 5 | |
| | Sync for errors | 7 | | 5 | |
| HW and Processing Chain | | 8 | | 5 | |
| | | 9 | | 5 | |
| | | 10 | | 5 | |
| | | 11 | | 5 | |
| | CoCo flag & interrupt | 12 | | 5 | ED:A, ED:D, S&D (2, 2, 1). PWH ok. 1 pt off per extra answer |
| | | 13 | | 5 | 1 pt off per extra answer. |
| | | 14 | | 5 | 1 pt off per extra answer. |
| | Automatic compare | 15 | | 5 | ED:A, ED:D, S&D (2, 2, 1). PWH ok. 1 pt off per extra answer |
| | Hardware average | 16 | | 5 | Ev. Det.: Analyze and/or Proc. Work/Handler. 1 pt off per extra answer. |
| | **560:** Compensation for errors in offset, gains | 17 | | 5 | DC:Q required. ED:A ok, PWH ok. 1 pt off per extra answer |
| | **560:** Self-calibration mode | 18 | | 5 | Flexible, looking for Trigger + multiple stages |
| | **560:** Sel. Voltage Reference | 19 | | 5 | DC:Q required. 1 pt off per extra answer. |

560: 130 points

460: 130 – 30 = 100 points

## Distribution of Responses to Questions 8 through 19

| Q. | | Triggering: Start ADC Conversion | Data Conv.: Sample | Data Conv.: Quantize | Ev. Det.: Analyze | Ev. Det.: Decide | Sched. & Dispatch | Proc. Work/ Handler |
|---|---|---|---|---|---|---|---|---|
| 8 | Linear successive approximation algorithm with up to 16-bit resolution | 7 | 40 | 113 | 4 | 4 | 2 | 4 |
| 9 | Configurable sample time and conversion speed/power | 0 | 113 | 74 | 0 | 0 | 2 | 2 |
| 10 | Output modes: Differential 16-bit, 13-bit, 11-bit, and 9-bit modes, single-ended 16-bit, 12-bit, 10-bit, and 8-bit modes | 0 | 13 | 109 | 2 | 1 | 3 | 11 |
| 11 | Single or continuous conversion, that is, automatic return to idle after single conversion | 45 | 20 | 17 | 1 | 8 | 50 | 6 |
| 12 | Conversion complete flag and interrupt | 2 | 3 | 1 | 35 | 71 | 93 | 54 |
| 13 | Software conversion trigger option | 101 | 2 | 2 | 1 | 0 | 9 | 4 |
| 14 | Selectable hardware conversion trigger with hardware channel select | 84 | 18 | 4 | 2 | 2 | 9 | 3 |
| 15 | Automatic compare with interrupt for less-than, greater-than or equal-to, within range, or out-of-range, programmable value | 0 | 1 | 6 | 109 | 106 | 69 | 43 |
| 16 | Hardware average function | 1 | 44 | 62 | 64 | 16 | 12 | 15 |
| 17 | ECE 560 Only: Compensation for errors in offset, plus-side gain and minus-side gain | 0 | 16 | 75 | 33 | 12 | 1 | 8 |
| 18 | ECE 560 Only: Self-Calibration mode to determine correction values for offset, plus-side gain and minus-side gain | 7 | 36 | 66 | 47 | 22 | 14 | 28 |
| 19 | ECE 560 Only: Selectable voltage reference: external or alternate | 7 | 58 | 71 | 1 | 1 | 1 | 3 |