

exam – **Embedded Software** – TI2726-B/TI2725-C

April 16, 2015 14.00 - 15.30

This exam (6 pages) consists of 60 True/False questions.
Your score will be computed as: $\max(0, \frac{\#correct}{60} - \frac{1}{2}) \times 2 \times 9 + 1$
It is **not** allowed to consult the book, handouts, or any other notes.

Instructions for filling in the answer sheet:

- You may use a **pencil** (erasures are allowed) or a **pen** (blue or black, **no** red, **no** strike outs).
 - Fill in the boxes **completely**.
 - Answer **all** questions; there is no penalty for guessing.
 - Do not forget to fill in your **Name** and **Student Number**, and to **sign** the form.
-

The following abbreviations are assumed to be known:

- FQS (Function Queue Scheduling)
- ISR (Interrupt Service Routine)
- RR (Round Robin)
- RRI (Round Robin with Interrupts)
- RTOS (Real-Time Operating System)

One system clock tick = 10 ms (unless stated otherwise).

We make use of the following definitions:

```
void delay(int ms) {  
    !! do some CPU computation to the number of ms milliseconds  
}  
  
void putchar(char c) {  
    while (!! UART tx buffer not empty)  
        ;  
  
    !! send c to UART tx buffer  
}  
  
void puts(char *s) {  
    !! write string s using putchar  
}
```

1. Embedded programming is more difficult than “classical” programming because of the thread-based programming model. true/false
2. The **Embedded software crisis** refers to the decrease in the number of manufactured embedded systems. true/false
3. A defining characteristic of embedded systems is the restricted memory size and processing power. true/false
4. Besides Finite State Machines other models of computation suitable for embedded systems include Publish/Subscribe and Discrete Events true/false
5. An interrupt is an asynchronous signal from hardware to indicate the need for processor attention. true/false
6. An embedded program can be coded as a finite state machine; the number of incoming transitions (arcs) into a state S must equal the number of outgoing transitions (arcs). true/false
7. VHDL is an ideal programming language for embedded systems as its synchronous model of computation supports multi-tasking at the hardware level. true/false
8. Using interrupts improves system response time. true/false
9. An interrupt service routine should save the context upon exit. true/false
10. To guarantee atomicity critical sections must be disabled. true/false

11.

```

int temp1, temp2;

void isr_buttons(void) // arrive here if a button is pressed
{
    temp1 = X32_PERIPHERALS[PERIPHERAL_TEMP1];
    temp2 = X32_PERIPHERALS[PERIPHERAL_TEMP2];
    ...
}

main() {
    ...
    while (!program.done) {
        X32_display = ((temp1 & 0xff) << 8) | (temp2 & 0xff);
        if (temp1 != temp2) {
            // shutdown plant
        }
    }
}

```

The above pseudo code suffers from the shared-data problem. true/false

12. An interrupt vector table contains the addresses of the interrupt service routines. true/false
13. An interrupt can **not** be serviced faster than the execution time of the shortest task in the system. true/false
14. The worst-case latency for servicing an interrupt is a combination of factors, including the time taken for higher priority interrupts. true/false

15. **Priority inversion** requires a minimum of 2 tasks of different priority and 1 semaphore to occur. true/false

16. A **deadly embrace** requires a minimum of 2 tasks and 2 semaphores to occur. true/false

17. Given the following pseudo code, which reads the current values of 4 different buttons and acts accordingly. The 4 buttons are all mapped to bits 0..3 of the button register. The buttons are already debounced.

```
void f1(void) { delay(1000); }
void f2(void) { delay(2000); }
void f3(void) { delay(3000); }
void f4(void) { delay(4000); }

void main (void) {
    while (1) {
        if (buttons & 0x01) f1();
        if (buttons & 0x02 ) f2();
        if (buttons & 0x04 ) f3();
        if (buttons & 0x08 ) f4();
        delay(1000);
    }
}
```

This code is an example of an RR architecture. true/false

18. When none of the buttons have been pressed, the longest time that button #3 must be pressed to activate f3() once is 1 second. true/false

19. When the system is in an arbitrary state, button #1 must be pressed at most 5 seconds to activate f1(). true/false

20. The shared-data problem can be solved through using mutexes. true/false

21. When a processor in an embedded system is powered up, interrupts are enabled to meet response-time requirements. true/false

22.

```
static volatile int count;

main () {
    ...
    count = 666;
    ...
}
```

Writing to the global variable count is atomic. true/false

23. On 8-bit processors the number of interrupt priorities is limited to 256 (2^8). true/false

24. The **primary** shortcoming of an RRI architecture is that critical sections must be used. true/false

25. An RRI architecture supports priority-based task scheduling. true/false

26. The response time to an external event in an FQS architecture depends on the longest task in the system. true/false

27. With an RRI architecture, a task associated with a high-priority interrupt is executed immediately after that ISR completes execution. true/false
28. Consider an alarm system that constantly monitors the digital output of several motion detector sensors in a house. If a breach is detected then an intermittent alarm sound is triggered.
- To guarantee a minimum response time an RRI or more advanced architecture must be used. true/false
29. A FQS architecture reduces to an RRI architecture when tasks are serviced in FIFO order. true/false
30. In an RTOS, tasks can be in state BLOCKED, READY or RUNNING.
- a task starts in the state RUNNING. true/false
31. Semaphores can be used for signaling between tasks. true/false
32. An ISR may change a task's status from RUNNING to READY. true/false
33. A semaphore used for guaranteeing mutual exclusive access to shared resources must be initialized to 1. true/false
34. A function can be made reentrant by means of a critical section, but then it may no longer be called by an ISR. true/false
35. A program running on an RTOS may create tasks dynamically at runtime.
- new tasks may only be spawned by the `main()` function. true/false
36. Context switching from one task to another is only slightly more expensive than an ordinary function call as the difference is that the stack pointer must be adjusted as well. true/false
37. An ISR may call the `OS_post()` routine, but not the `OS_pend()` routine. true/false
38. When upgrading to an RTOS, signaling between ISRs and tasks may still be done through flags residing in global memory. true/false
39. In the implementation of the `OS_Pend()` primitive, the RTOS first switches the state of the current task to BLOCKED, and then looks for a task in the READY queue.
- if the READY queue is empty the program is deadlocked and may be aborted. true/false
40. Local variables can be used at will without creating a shared-data problem. true/false
41.

```
int f (int x) {
    disable_int();

    !! touch some global variables
    !! do some processing
    !! call some functions

    enable_int();
}
```
- Function `f()` that disables/enables interrupts on entry/exit fails to address the shared-data problem when calling itself recursively. true/false

42. The 'alternating buffers' technique addresses the shared-data problem by copying the data from the in- to the out-buffer instead of passing a pointer. true/false
43. An RTOS usually provides two types of delay functions: polling-based and timer-based.
- timer-based delays are the most accurate. true/false
44. Assume that one system clock tick = 10 ms.
- Calling the function `OSTimeDly(4)` causes a delay of exactly 40 ms. true/false
45. The **heartbeat timer** is a single hardware timer an RTOS is using as base for all timings. true/false
46. When using an RTOS the `delay()` function may not be used because the hardware timer is used to implement time slicing. true/false
47. To address the shared-data problem, many RTOSs provide communication primitives like queues, mailboxes, and pipes.
- a common pitfall is that they allow pointers to be passed from one task to another. true/false
48. A key principle of RTOS-based design is that short interrupt routines are needed for a responsive system true/false
49. Aborting tasks is nontrivial because a task may hold resources (e.g., a semaphore) when being destroyed. true/false
50. Time-slicing should be avoided in an RTOS because it extends the *deadly embrace* problem to tasks of equal priority. true/false
51. In an RTOS each task requires its own stack space. true/false
52. Printing from an ISR is considered bad practice as the driver resides in the RTOS. true/false
53. Time slicing between ISRs is common practice in embedded systems. true/false
54. A logic analyzer is preferred to an in-circuit emulator because it can be used with any type of processor. true/false
55. When developing code for an embedded system, the software can be structured into HW-dependent and HW-independent code.
- Doing so makes debugging HW-independent code feasible on the host platform true/false
56. Debugging through scripting test scenarios has limited use as only one interrupt can be triggered at the exact same time. true/false
57. When debugging code for a distributed sensor network, collecting the (debug) output of the nodes can be arranged in different ways [Beutel:2009].
- **offline** sniffing requires logging facilities on the sniffer nodes. true/false
58. A large study of outdoor sensor-network deployments has shown that the most underestimated problem has been the water-proof packaging of the base station. true/false

- 59.** Given is the following RTOS (pseudo) code. T1 has the highest priority, the time for `puts` and context switching is negligible:

```
void T1(void) {
    while (1) {
        puts("1 ");
        OSTimeDly(10);
    }
}

void T2(void) {
    while (1) {
        puts("2 ");
        OSTimeDly(10);
    }
}
```

The display shows the sequence "2 1 2 1 2 1 2 1 2 ..."

true/false

- 60.** When we replace the `OSTimeDly(10)` call with a `delay(10)` call, the display will show the sequence "1 1 1 1 1 1 1 1 1 ..."

true/false