Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

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

January 28, 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 event-based programming model. true/false

2. A defining characteristic of embedded systems is the lack of an interrupt controller. true/false

3. The **Embedded software crisis** refers to the lack of correct code for the increasing number of embedded systems. true/false

4. Despite advances in software engineering practices, as a rule of thumb, embedded software contains 1-10 bugs per million lines of code. true/false

5. An embedded program can be coded as a finite state machine where interrupts trigger state transitions. true/false

6. An interrupt is a synchronous signal form hardware to indicate the need for processor attention. true/false

7. Besides Finite State Machines other models of computation suitable for embedded systems include Publish/Subscribe and Recursion. true/false

8. Since disabling interrupts increases interrupt latency, several alternative methods have been developed for dealing with shared data. The Alternating Buffers method is suited for handing data from an ISR to a task.

```
static int tempA[2], tempB[2];
static bool useB = FALSE;

void interrupt readTemp() {
   if (useB) {
      tempA[0]= ...;
      tempA[1]= ...;
   } else {
      tempB[0]= ...;
      tempB[1]= ...;
   }
}

void main(void) {
   while (TRUE) {
      if (useB)
         if (tempB[0]!=tempB[1]) ...  ;
      else
         if (tempA[0]!=tempA[1]) ...  ;
      useB = !useB;
   }
}
```

The code for toggling the `useB` flag should be in the main task (not the ISR) as shown above. true/false

9. Using interrupts improves context switch times. true/false

10. An interrupt service routine should restore the context upon exit. true/false

11. To guarantee atomicity task switching must be disabled. true/false

12. A low-priority ISR can be interrupted by a high-priority ISR. true/false

13. The shared data problem can be solved through using semaphores. true/false

14. When a processor is powered up, interrupts are disabled until further notice. true/false

15. **Priority inversion** requires a minimum of 3 tasks of different priority and 1 semaphore to occur. true/false

16. An interrupt vector points to a table with interrupt routines. true/false

17. An interrupt can **not** be serviced faster than the time needed to save the context of code running on the processor. true/false

18. Mutual exclusive access can also be accomplished by disabling interrupts, which has the advantage of faster system response compared to using RTOS primitives like semaphores and mutexes. true/false

19.

```
int panic = 0;

void isr_buttons(void) // arrive here if a button is pressed
{
   int temp1 = X32_PERIPHERALS[PERIPHERAL_TEMP1];
   int temp2 = X32_PERIPHERALS[PERIPHERAL_TEMP2];
   if (temp1 != temp2) {
      panic = 1;
   }
   ...
}

main() {
   ...
   while (!program_done) {
      // some lengthy calculations and control commands
      if (panic) {
         // shutdown plant
      }
   }
}
```

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

20. The worst-case latency for servicing an interrupt is a combination of factors, including the longest period of time in which interrupts are disabled. true/false

21. A **deadly embrace** requires a minimum of 3 tasks and 2 semaphores to occur. true/false

22.

```
static volatile int count;

main () {
   ...
   int val = count;
   ...
}
```

Reading the value of the global variable `count` is atomic. true/false

23. 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 RRI architecture. true/false

24. When none of the buttons have been pressed, the longest time that button #3 must be pressed to activate f3() once is 4 seconds. true/false

25. When the system is in an arbitrary state, button #1 must be pressed at most 9 seconds to activate f1(). true/false

26. Given is the following RTOS (pseudo) code with priority T1 $>$ T2.

```
void T1(void) {
   while (1) {
      OS_Pend(sem1); // event #1 may unblock any time
      f(1);
      OSTimeDly(1);
   }
}

void T2(void) {
   while (1) {
      OS_Pend(sem2); // event #2 may unblock any time
      f(-1);
      OSTimeDly(3);
   }
}

void f(int i) {
   OS_Pend(mutex);
   counter = counter + i ; // modify some global counter
   OS_Post(mutex);
}
```

This code suffers from a data sharing problem. true/false

27. If the order of events is #1, #2, #1, #2, #1 and they occur within 10 ms from each other, then the final value of the counter will be increased by 1. true/false

28. The function `f()` is reentrant true/false

29. The **primary** shortcoming of an FQS architecture is that all tasks have the same priority. true/false

30. An RR architecture does **not** support priorities. true/false

31. With an FQS architecture, a task signaled by an ISR is executed immediately after that ISR completes execution. true/false

32. 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.
- That alarm system can be implemented with an RR architecture. true/false

33. An RTOS architecture is most robust to code changes. true/false

34. In an RTOS, tasks can be in state BLOCKED, READY or RUNNING.
- A task can transition directly from READY to RUNNING. true/false

35. An ISR may change a task's status from RUNNING to BLOCKED. true/false

36. An ISR can signal a task by operating a semaphore. true/false

37. A reentrant function may use hardware only in an atomic way. true/false

38. A reentrant function may **not** reference variables labeled `extern`. true/false

39. A semaphore used for condition synchronization must be initialized to zero. true/false

40. An ISR must **not** invoke an RTOS function that may block. true/false

41. A function can be made reentrant by temporarily disabling interrupts, but additional bookkeeping is required as simply enabling interrupts on exit may cause errors. true/false

42. An ISR may call the `OS_pend()` routine, but not the `OS_post()` routine . true/false

43. When using an RTOS signaling between ISRs and tasks must be done by calling appropriate RTOS primitives. true/false

44. A program running on an RTOS may create tasks dynamically at runtime.
- the program ends as soon as the `main()` function returns. true/false

45. The **heartbeat timer** is a single hardware timer an RTOS is using to verify that the system is still progressing (i.e. not deadlocked). true/false

46. An RTOS usually provides two types of delay functions: polling-based and timer-based.
- polling-based delays are the most accurate. true/false

47. Assume that one system clock tick = 10 ms.
- Calling the function `OSTimeDly(5)` causes a delay between 40 and 50 ms. true/false

48. Time-slicing should be avoided in an RTOS because it makes the response time of tasks less predictable. true/false

49. A key principle of RTOS-based design is that the separation of concerns, by splitting code amongst several tasks, improves the overall throughput. true/false

50. Creating and destroying tasks dynamically is somewhat problematic because the RTOS must disable interrupts for too long. true/false

51. Tasks in an RTOS are often structured as state machines with states stored in private variables and ISRs advancing the state machine. true/false

52. It is recommended to use just the minimum necessary functionality from an RTOS. true/false

53. Tasks can share the same stack as mutual exclusion allows only one task to execute a critical section. true/false

54. A logic analyzer is preferred to an in-circuit emulator because it can monitor the internal memory bus of (most) modern micro controllers. true/false

55. When developing code for an embedded system, the software can de structured into HW-dependent and HW-independent code.
- Doing so makes debugging HW-independent code feasible on the target platform true/false

56. Debugging through scripting test scenarios can **not** be used to test HW-dependent code. true/false

57. A large study of outdoor sensor-network deployments [Beutel:2009] has shown that the two most underestimated problems have been the water-proof packaging of the sensor nodes and the provision of a reliable base station. true/false

58. When debugging code for a distributed sensor network, collecting the (debug) output of the nodes can be arranged in different ways.
- **online** sniffing requires logging facilities on the sensor nodes themselves. 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 ”1 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 ouput of the print statements will be displayed in a random order. true/false