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

exam – **Embedded Software** – TI2726-B
April 19, 2017 13.30 - 15.00

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:

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

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. A defining characteristic of embedded systems is the usage of a rich user interface. true/false

3. The **Embedded software crisis** refers to the decrease in the number of manufactured embedded systems. true/false

4. Several models of computation for embedded systems are described in [Lee:2002].
- Process Networks are primarily used to describe concurrency at the hardware level. 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 an asynchronous signal from hardware to indicate the need for processor attention. true/false

7. Finite State Machines can be coded in VHDL.
- An advantage of doing so is that it results in lower interrupt latency as less context (e.g., registers) need to be saved and restored. true/false

8. The C language does not contain a built-in type to represent booleans.
- True and False are handled as numeric values 1 and 0, respectively. true/false

9. Memory allocated by the `malloc()` function is located on the data heap above the code. true/false

10.

```
typedef void *resolve(void *old, void *new);
```

The definition above declares `resolve` as a pointer to a function that takes two arguments of type `void *` and returns a void as result. true/false

11. GDB is programming tool that aids memory debugging by executing a program in a safe environment. true/false

12.

```
int main(void)
{
   int c;
   statefp state = before;
   while((c = getchar()) != EOF) {
      state = (statefp) (*state)(c);
   }
   return 0;
}
```

The above driver loop for a FSM follows a round-robin architecture. true/false

13. Specifying the type of `statefp` is difficult in C because it is recursive and types cannot be referenced before being fully defined.
- This explains the need for an explicit type cast in the body of the while loop. true/false

14. Using interrupts improves task response time. true/false

15. Disabling interrupts guarantees atomicity of the code until they are enabled again. true/false

16. An interrupt service routine should save the context upon entrance. true/false

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

18.

```
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

19. The shared-data problem can be solved by storing data in non-volatile memory. true/false

20. Critical sections can be guarded by disabling and enabling interrupts.
- interrupts arriving during such a critical section are buffered and handled upon exit. true/false

21. When a processor is powered up, the state of the interrupt controller needs to be initialized before the RTOS can be invoked. true/false

22. **Priority inversion** occurs when the `volatile` and `static` keywords are wrongly used inside a task or interrupt. true/false

23.

```
static volatile int count;

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

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

24. 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

25. The number of interrupts is limited by the number of GPIO pins on the processor. true/false

26. An interrupt vector table contains the code of the interrupt service routines. true/false

27. 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

28. An RR architecture supports priority-based task scheduling. true/false

29. With an RRI architecture, the execution of a task associated with a high-priority interrupt may be delayed by other tasks in the system. true/false

30. An FQS architecture has a smaller memory footprint than an RTOS as it needs only one stack. true/false

31. 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

32. The response time to an external event in an FQS architecture is deterministic and depends solely on the length of the ISR. true/false

33. When detecting a car crash an airbag should not be inflated instantly.
- An RTOS provides functionality to support such delayed actions. true/false

34. When upgrading to an RTOS, signaling between ISRs and tasks may still be done through flags residing in global memory. true/false

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

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

37. A reentrant function may only be used by one task at a time — true/false

38. Semaphores can be used for signaling between tasks. — true/false

39. Tasks may call the `OS_pend()` routine, but not the `OS_post()` routine . — true/false

40. A program running on an RTOS may create tasks dynamically at runtime.
- the program ends once `main()` and all spawned tasks have finished. — true/false

41. 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 RTOS starts a watchdog timer to guard for a potential deadlock. — true/false

42. 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);
   }
}

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

void f(int i) {
   delay(10); // do some computation
   counter = counter + i ; // modify some global counter
   printf("%d\n", counter) ; // print result
}
```

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

43. If `counter` is set to 15 when event 2 occurs, and event 1 follows 13 ms later, then the first value printed is 15. — true/false

44. If the call to `delay` is replaced with `OSTimeDly` the output will be different. — true/false

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

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

47. To address the shared-data problem, many RTOSs provide communication primitives like queues, mailboxes, and pipes.
- a common advantage is that they allow pointers to be passed from one task to another. — true/false

48. The advantage of queues over pipes is that messages/items can be of variable length. — true/false

49. Even when an RTOS is aware of which task is using which semaphore, it cannot prevent deadlock. true/false

50. Tasks in an RTOS are often structured as state machines with states stored in private variables and messages in their queues acting as events. true/false

51. The memory footprint of a program grows linearly with the number of tasks. true/false

52. Printing from an ISR is considered a good practice as no other debugging techniques are available. true/false

53. Time slicing between tasks of equal priority is to be avoided as the response time of individual tasks is comprised. true/false

54. A semaphore S used by task A must be initialized by A. true/false

55. Aborting tasks is nontrivial because a task may hold resources (e.g., a semaphore) when being destroyed. true/false

56. 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

57. Code coverage tools help in thorough testing.
- a 100% coverage implies a bug-free program. true/false

58. Even on embedded devices **without** a display, the assert macro is a useful debugging aid. true/false

59. A large study of outdoor sensor-network deployments [Beutel:2009] has shown that the water-proof packaging of the base station is key to establishing a reliable connection to the back bone. true/false

60. When debugging code for a distributed sensor network, collecting the (debug) output of the nodes can be arranged in different ways.
- A **wireless** testbed requires physical instrumentation (i.e. wiring) of the sensor node. true/false