This exam (6 pages) consists of 60 True/False questions. Your score will be computed as: \( \text{max}(0, \frac{\# \text{correct}}{60} - \frac{1}{2}) \times 2 \times 9 + 1 \)

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)
- UART (Universal Asynchronous Receiver Transmitter)

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

We make use of the following definitions:

```c
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 lack of support for recursion.  
false

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

3. Because embedded software engages the physical world, it has to embrace time and other non-functional properties, which requires a view that is significantly different from the prevailing abstractions in computation.  
true

4. Finite State Machines can be coded in VHDL.  
- An advantage of doing so is that it results in a fast and predictable process executing on dedicated hardware.  
true

5. Interrupts cannot only be generated by hardware, but also by software.  
- A software interrupt is a synchronous signal to indicate the need for a change in the execution flow.  
true

6. An embedded program can be coded as a finite state machine.  
- When for every state S the number of incoming transitions (arcs) equals the number of outgoing transitions (arcs), the code is free of deadlocks.  
false

7. Besides Finite State Machines other models of computation suitable for embedded systems include Symbolic Execution and Discrete Events.  
false

8. The size of an int is architecture dependent, but defined to be larger than a short.  
false

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

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

11. 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.  
false

12. Specifying the type of statefp is difficult in C because it is recursive and types cannot be referenced before being fully defined.  
true

13. GDB is programming tool that provides controlled execution of an executable.  
- it also provides post mortem inspection when a core file is generated.  
true

true

15. An interrupt service routine should restore the context upon exit.  
true
16. To guarantee atomicity critical sections must be disabled.  
   \textit{false}

17. An ISR can \textbf{not} be interrupted by another ISR.  
   \textit{false}

18. When a processor is powered up, the state of the interrupt controller needs to be initialized  
    before the RTOS can be invoked.  
   \textit{false}

19. \begin{verbatim}
    static int iSeconds, iMinutes;
    void interrupt vUpdateTime(void)
    {
      ++iSeconds;
      if (iSeconds>=60) {
        iSeconds=0;
        ++iMinutes;
      }
    }
    long lSeconds(void)
    {
      disable();
      int now = iMinutes*60+iSeconds;
      enable();
      return(now);
    }
\end{verbatim}

The above pseudo code correctly dis-/enables the interrupts to solve the shared-data  
problem.  
   \textit{true}

20. An interrupt vector table contains the code of the interrupt service routines.  
   \textit{false}

21. Given the following pseudo code, which reads the current values of 3 different buttons  
    and acts accordingly. The 3 buttons are all mapped to bits 0..2 of the button register. The  
    buttons are already debounced.\begin{verbatim}
    void f1(void) { delay(1000); }
    void f2(void) { delay(2000); }
    void f3(void) { delay(3000); }

    void main (void) {
      while (1) {
        if (buttons & 0x01) f1();
        delay(1000);
        if (buttons & 0x02 ) f2();
        delay(1000);
        if (buttons & 0x04 ) f3();
      }
    }
\end{verbatim}

This code is an example of an RR architecture.  
   \textit{true}

22. When none of the buttons have been pressed, the longest time that button #3 must  
    be pressed to activate f3() once is 4 seconds.  
   \textit{false}

23. When the system is in an arbitrary state, button #1 must be pressed at most 10 seconds  
    to activate f1().  
   \textit{false}

24. The worst-case latency for servicing an interrupt is a combination of factors, including  
    the time taken for higher priority tasks.  
   \textit{false}
25. The number of interrupts is limited by the number of GPIO pins on the processor.  
   false

26. Mutual exclusive access can also be accomplished by disabling interrupts, which has the  
   advantage of faster context switching compared to using RTOS primitives like semaphores  
   and mutexes.  
   false

27. **Priority inversion** requires a minimum of 3 tasks of different priority and 3 semaphores  
   to occur.  
   false

28. The **primary** shortcoming of an RRI architecture is that all tasks have the same priority.  
   true

29. An FQS architecture supports priority-based ISRs.  
   true

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

31. An RR architecture is most robust to code changes.  
   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

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

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

35. Semaphores can be used for signaling between ISRs.  
   false

36. A reentrant function may **not** reference variables labeled `extern`.  
   false

37. A semaphore used for guaranteeing mutual exclusive access to shared resources must be  
   initialized to 1.  
   true

38. A high-priority task must **not** invoke an RTOS function that may block.  
   false

39. The ‘alternating buffers’ technique addresses the shared-data problem by having the  
   RTOS control when to switch between buffers.  
   false

40. 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 processor may be put into sleep mode to save energy  
   when idling.  
   true
41. int f (int x) {
  disable_int();

  // read some global variables
  // do some processing, call some functions
  // write some global variables

  enable_int();
}

Function \textit{f}() disables/enables interrupts to address the shared-data problem.
- However, when \textit{f}() calls itself recursively, it is no longer reentrant. \textit{true}

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

\begin{verbatim}
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
}
\end{verbatim}

The function \textit{f}() is reentrant. \textit{false}

43. If \textit{counter} is set to 15 when event 2 occurs, and event 1 follows 3 ms later, then the first value printed is 16. \textit{true}

44. If the call to \textit{delay} is replaced with \textit{OSTimeDly} the output will be different. \textit{false}

45. An RTOS usually provides two types of delay functions: polling-based and timer-based.
- polling-based delays are more efficient as other tasks can run while the caller is waiting for the specified time to pass. \textit{false}

46. Assume that one system clock tick = 10 ms.
- Calling the function \textit{OSTimeDly} \( 5 \) causes a delay between 40 and 50 ms. \textit{true}

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. \textit{false}

48. The advantage of queues over pipes is that messages/items can be of variable length. \textit{false}

49. Even when an RTOS is aware of which task is using which semaphore, it cannot prevent deadlock. \textit{true}
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

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

52. Printing from an ISR is to be avoided except when the RTOS provides a reentrant primitive to do so.  
   true

53. Time slicing between tasks of equal priority is to be avoided as it compromises the predictability of their response times.  
   true

54. A semaphore S used by task A must be initialized before A is created.  
   false

55. It is recommended to use just the minimum necessary functionality from an RTOS.  
   true

56. Code coverage tools help in thorough testing.  
   - A 100% coverage implies a bug-free program.  
   false

57. A logic analyzer is preferred to an in-circuit emulator because it is easier to install; not all signals need to be connected.  
   true

58. Debugging through scripting test scenarios can only be used to test HW-independent code.  
   true

59. A large study of outdoor sensor-network deployments [Beutel:2009] has shown that the most underestimated problem has been securing the power supply of the sensor nodes.  
   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 no physical instrumentation (i.e. wiring) of the sensor nodes.  
   false