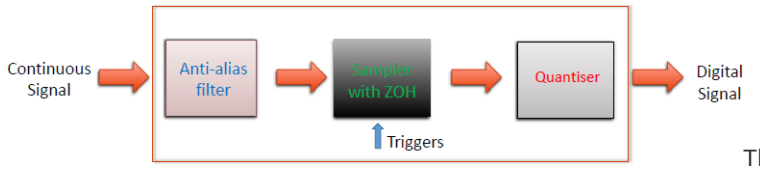
Question 1

Not answered

Marked out of 2.0

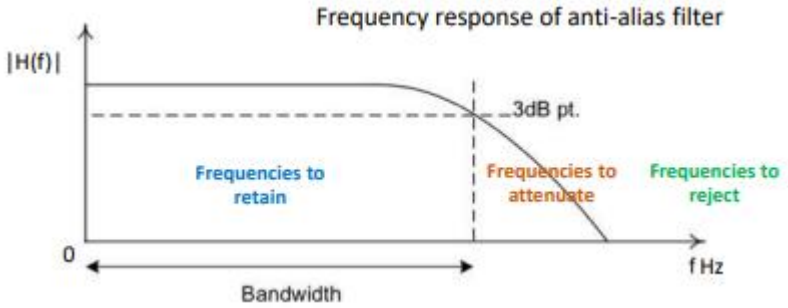
ADC diagram



The figure shows the diagram of a typical analogue-to-digital converter (ADC). Explain briefly the function of the anti-alias filter.

Answer 1)

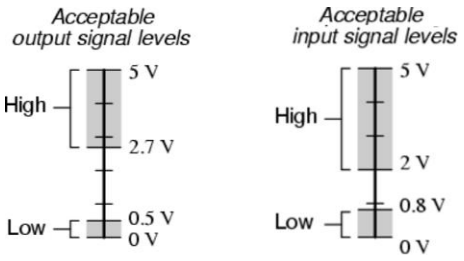
- According to to Nyquist-Shannon Theory, to fully reconstruct the original wave form from samples, we must sample at a frequency f much greater than 2*W. Where W is the highest input signal frequency.
- When the sampling frequency f < 2*W, aliasing will occur
- The function of the anti-aliasing filter would prevent the aliasing when f is not high enough, as it's a low-pass filter that 'band-limits' the input signal
- It does this by reducing the amount of energy that has risen at the point of aliasing as shown in diagram below.



Question 2

Not answered

Marked out of 3.0



The figure below shows the voltage levels for logic-high and logic-low (both shaded) of a particular device technology.

Find the logic-high noise margin and logic-low noise margin.

Answer 2)

- High level noise margin in TTL (transistor - translate logic) = 2.7 - 2 = 0.7V
- Low-level noise margin in TTL (transistor - translate logic) = 0.7 - 0.5 = 0.2V

Question 3

Not answered

Marked out of 4.0

A 16-bit count up/down timer is used to measure intervals up to 100 ms.

Suppose the MCU clock rate is 20 MHz and the timer clock prescaler value is an 8 bit unsigned integer.

What is the best time resolution possible for the interval measurements and what is the value of the prescaler to achieve that?

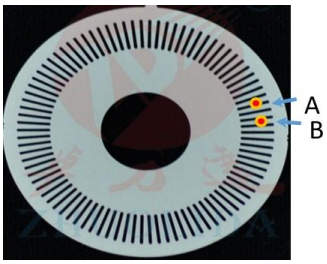
Answer 3)

- What we know is: prescaler = k (to find out), clock_ rate = 20MHz, Maximised time resolution = 0.1s, timer = 16 bits
- Time resolution = prescaler / clock_rate = k / 20MHz
- Maximised time resolution = 0.1 = [2^16] * [Time resolution] = [2^16] * [k / 20MHz]
- Rearrage to find the prescaler k = [0.1 * 20MHz] / [2^16] = 30.52Hz
- Best time resolution = prescaler / clock_rate = 30.52Hz / 20MHz = 1.52600×10^-6s = 1.53us
- Therefore best time resolution = 1.53us, with a prescaler of 30.52Hz

Question 4

Not answered

Marked out of 4.0



Suppose the quadrature encoded disk shown below has 100 slots and it rotates at 5 revolutions per second. A pair of optical sensors (sensors A and B) are used to determine the distance and direction of the disk's rotation.

Assuming that the two sensors are connected to GPIO pins in the same port, what is the minimum time between two interrupts generated by changes in the sensor signals?

Answer 4)– unsure if on the right track

- What we know: frequency f = 5Hz
- 100 slots * f = 100 * 5 = 500 slots rotating per second
- 1/K = 500/second = 500/1000ms = 0.5 slots rotating per millisecond
- K = 1/0.5 = 2 millisecond will have a slot
- Therefore minimum time between two interrupts = 2 * 2 milliseconds = 4 ms

Question 5

Not answered

Marked out of 2.0

The interrupt service routines are now removed and polling is used instead to detect changes in the sensor signals for the quadrature encoded disk used in question 4 and shown below.

Calculate the minimum polling frequency required to correctly capture the disk's rotation, assuming the number of slots (100) and rate of rotation (5 revolutions per second) have not changed. State any further assumptions as required.

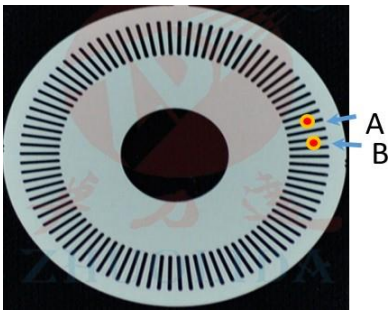
Answer 5)– unsure if on the right track

- 1/4ms = 1/0.004s = 250 Hz

Question 6

Not answered

Marked out of 4.0



A pulse-width modulated (PWM) signal has a period of 200 ms, 45% duty cycle, a minimum voltage of 0 V and a maximum voltage of 3 V.

Find the average voltage of this PWM waveform.

Answer 6)

- 3 V * 0.45 = 1.35V average

Question 7

Not answered

Marked out of 1.0

A different PWM signal from the one in question 6 has a period of 500 ms, 60% duty cycle, minimum voltage of 0 V and maximum voltage of 5 V.

Calculate the signal's root mean square (RMS) voltage.

Answer 7)

- RMS of square wave is equal to the amplitude, so 5-0 = 5.

Question 8

Not answered

Marked out of 2.0

Explain briefly what the context for an interrupt service routine (ISR) call is.

Answer 8)

- The context is the information that is needed for returning back to the main program from the ISR. The information includes pc, PSR, lr, r0-r3, r12.

Question 9

Not answered

Marked out of 5.0

A microcontroller with a 4 MHz clock is responding to two interrupts A and B, using interrupt service routines (ISRs) ISR_A and ISR_B, respectively. The two interrupts have the same priority.

Interrupt A occurs no more frequently than once every 150 us, while Interrupt B occurs no more frequently than once every 200 us.

The machine instructions of the processor take from 4 to 10 CPU clock cycles to execute.

It takes 5 clock cycles to store the context for ISR_A. It also takes 5 clock cycles to store the context for ISR_B.

Measurements have indicated that the worst case execution time for each ISR (including the prologue and epilogue) is 7.5 us (ISR_A) and 6 us (ISR_B).

What are the worst case interrupt response times for servicing interrupts A and B in clock cycles, based on the information given above?

Show your working and state the assumptions you make when answering.

Answer 9)

- Interrupt A: once every 150us
- Interrupt B: once every 200us
- Instructions take 4-10 clocks to execute.
- Takes 5 clocks to store context.
- WC ISR A: 7.5us
- WC ISR B: 6us
- Interrupt response = latency + context switch time

Question 10

Not answered

Marked out of 6.0

Suppose a free-running timer has been configured such that it can generate timer interrupts periodically.

Explain briefly how the following code realizes a time-triggered scheduler and how the code may be modified to realize task prioritization.

Hint:

```
void timerInterruptHandler(void)
```

is the interrupt service routine (ISR) for servicing the periodically generated timer interrupts;

```
void main(void)
```

is the main function; and

```
scheduled_task[i].delay
```

is declared as an unsigned integer.

```
void timerInterruptHandler (void)
{
    for (uint8_t i = 0; i < N_TASKS; i++)
    {
        if (scheduled_task[i].delay == 0)
        {
            scheduled_task[i].ready = true;
            scheduled_task[i].delay = scheduled_task[i].period;
        }
        else
        {
            scheduled_task[i].delay--;
        }
    }
}

void main (void)
{
    while (true)
    {
        for (uint8_t i = 0; i < N_TASKS; i++)
        {
            if (scheduled_task[i].ready)
            {
                // Call the handler of task[i]
                ...
                scheduled_task[i].ready = false;
            }
        }
    }
}
```

Answer 10) Explain briefly how the following code realizes a time-triggered scheduler

- It divides the program task up so when a task is flagged to run, it will. Hence this is a kernal that schedules tasks.
- The while(true) loop is the kernal of this operating system
- The kernal is a time-triggered scheduler as the task is time-divided up, with multiplexing of the MCU (time-division multiplexing)
- You can see it is time divided up with the code having delay counter being set to 0, or the task period, performing the task at controlled time.

Answer 10) how the code may be modified to realize task prioritization.

- Use an interrupt-triggered scheduler to prioritize foreground task over the background task.

Interrupt Triggered Scheduler Code:

volatile boolean flagDeviceA = false; . . . ; flagDeviceZ = false;

void deviceAIntHandler (void)

{

/* Handle Device A I/O, set flag */

```
}

. . . .

void deviceZIntHandler (void)
{
/* Handle Device Z I/O, set flag */
}

void main (void)
{
        . . . .
        while (true)
        {
                if (flagDeviceA) {
                // Process data to or from Device A and reset flag
                }
                . . . .
                if (flagDeviceZ) {
                 // Process data to or from Device Z and reset flag
                }
        }
}
```

Question 11

Not answered

Marked out of 3.0

Adam Ant uses a Count Event in Code Composer Studio to measure the number of elapsed clock cycles between breakpoints in his code. An excerpt of the code is shown below.

```
1  { ... }
31 #define BUF_SIZE 10
32 static circBuf_t g_inBuffer;
33
34 { ... }
48 void SysTickIntHandler(void)
49 { ... }
56
57 void ADCIntHandler(void)
58 { ... }
72
```