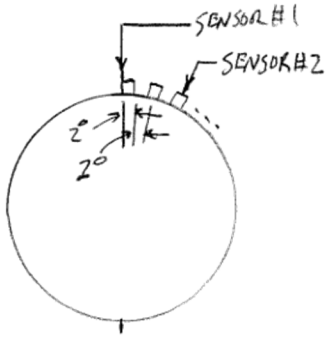
1. Write up the requirements for a rotary relative position encoder than has one degree (or somewhat better) of resolution and can keep track of the direction of rotation. Specify the number of teeth and the location of sensors. Make an illustrative drawing showing any important details. You may abbreviate repetitious elements of the situation.



NOT DRAWN TO SCALE

I WILL USE A QUADRATURE ENCODER DESIGN

RESOLUTION = $\frac{360^\circ}{4n}$ WHERE n = # OF TEETH

TO GET 1° OF RESOLUTION $1^\circ = \frac{360^\circ}{4n} \rightarrow n = \frac{360^\circ}{4^\circ} = 90$ TEETH

EACH TOOTH NEEDS TO BE 2° WIDE. SIMILARLY, EACH GAP NEEDS TO BE 2° WIDE

TWO SENSORS ARE NEEDED.

LET THE LOCATION OF SENSOR #1 BE USED TO DEFINE THE READ LINE.

SENSOR #2 MUST BE LOCATED $4^\circ k + 1^\circ$ FROM SENSOR #1

SET UP EACH SENSOR SO THAT IT WILL GENERATE AN INTERRUPT ON BOTH RISING AND FALLING TRANSITIONS. NOTE THAT WHEN AN INTERRUPT HAPPENS THE OTHER SENSOR WILL BE OVER THE CENTER OF A TOOTH OR GAP.

THE ISR FOR EACH SENSOR MUST SAMPLE EACH SENSOR BEFORE THE WHEEL CAN TURN 1/2 TOOTH OR 1°. (SO THAT THE SENSOR VALUES CANNOT CHANGE BETWEEN INTERRUPT REQUEST AND INTERRUPT SERVICE.) SUPPOSE A SENSOR READS LOGIC-1 WHEN OVER A TOOTH

SENSOR #1 ISR: IF SENSOR #1 = SENSOR # 2 ROTATION IS CCW, INC. A COUNTER
IF SENSOR #1 ≠ SENSOR # 2 ROTATION IS CW, DEC. COUNTER

8/8

SENSOR #2 ISR: IF SENSOR #2 ≠ SENSOR #1 ROTATION IS CCW, INC. A COUNTER
IF SENSOR #2 = SENSOR #1 ROTATION IS CW, DEC. A COUNTER

TO COUNT UP TO 90 FULL TURNS REQUIERS $(360^\circ/\text{TURN})\left(\frac{1\text{ COUNT}}{1^\circ}\right)(90\text{ TURNS}) = 32400$ COUNTS

$\log_2(32400) = 14.98$ ∴ THE COUNTER MUST HAVE 15 BITS

PROBABLY A 16 BIT COUNTER IN TWOS COMPLEMENT CODE WILL BE USED TO TRACK UP TO ±90 ROTATIONS.

2. (This problem is not assigned.)

3.17 **Interrupt Density and Interval Constraints**. Consider a design which has four sources of interrupts, all of which can be active at the same time. The sources are identified by numbers 1, 2, 3, and 4, with 1 representing the highest priority source and 4 the lowest. The corresponding interrupt service routines take T_1=31, T_2=29, T_3=37, and T_4=43 µs, respectively. The minimum time between interrupts from the same source is TP_1=120, TP_2=200, TP_3=150, TP_4=1000 µs, respectively. For reliable operation, each source must be serviced before its next interrupt occurs.

a.) Will operation be reliable? Assume further interrupts are disabled for the duration of each interrupt service routine. Explain your answer.

Interrupt Density = 31/120 + 29/200 + 37/150 + 43/1000
Interrupt Density = 0.693 < 1.0 which is OK.

Now find the interrupt interval constraint inequalities.

N(i,x) = maximum number of times interrupt x might be called during the maximum amount of time that interrupt i can be held in latency. (Interrupt x has higher priority than i)

$N(i,x) = \text{ceil}\{(T_{Pi} - T_i)/T_{Px}$ Note: N(i,i) = 1 always.

N(1,1) = 1	N(2,1) = 2	N(3,1) = 1	N(4,1)= 8
	N(2,2) = 1	N(3,2) = 1	N(4,2)= 5
		N(3,3) = 1	N(4,3)= 7
			N(4,4)= 1

T_{i+} = Max(lower priority T_i or longest critical region, or longest instruction)

T_{1+} = 43 us
T_{2+} = 43 us
T_{3+} = 43 us
T_{4+} = ? Since this is not specified I will leave this as a variable (or you could assume 0).

For Interrupt 1
Is $T_{1+} + N(1,1)T_1 < T_{P1}$?
43 + (1)(31) <120 ? (all units are µs)
74 <120 Therefore interrupt 1 is OK

4/4

For Interrupt 2
Is $T_{2+} + N(2,2)T_2 + N(2,1)T_1 < T_{P2}$?
43 + (1)(29) + (2)(31) < 200 ? (all units are µs)
134 < 200 Therefore interrupt 2 is OK

For Interrupt 3
Is $T_{3+} + N(3,3)T_3 + N(3,2)T_2 + N(3,1)T_1 < T_{P3}$?
43 + (1)(37) + (1)(29) + (1)(31) < 150 ? (all units are µs)
140 < 150 Therefore interrupt 3 is OK

For Interrupt ~~3~~ 4
Is $T_{4+} + N(4,4)T_4 + N(4,3)T_3 + N(4,2)T_2 + N(4,1)T_1 < T_{P4}$?
T_{4+} + (1)(43) + (7)(37) + (5)(29) + (8)(31) < 1000 ? (all units are µs)
T_{4+} + 695 < 1000 Therefore interrupt 3 is OK
if T_{4+} < 305 µs

Conclusion: **Operation will be reliable**. (Assuming T_{4+} < 53 µs, which number is derived from the constraint on interrupt 3.)

3.17 continued. . .

b.) If the mainline routine is to be permitted to include critical regions, during which interrupts are temporarily turned off, then how long can these last and still maintain reliable operation?

1/1

Observe in part (a) that the constraint on interrupt interval 3 is the tightest, with only 10 μs margin. If T_{4+} was 53 μs then this would increase T_{3+} to 53 μs too. That would be 10 μs more. Thus **the critical region (which will set T_{4+} must be less than 53 μs**.

c.) For case (a), given critical regions of no longer than 13 μs, then how much could T_1 be lengthened and still ensure reliable operation.

If critical regions are less than 13 μs, then T_{4+} = 13 μs. Now consider the role of T_1 in the various interrupt interval constraints.

For Interrupt 1
If T_1 = 0 there would be is 77 μs margin and T_1 could be called 1 time,
leaving a maximum for T_1 of 77 μs.

For Interrupt 2
If T_1 = 0 there would be is 128 μs margin and T_1 could be called 2 times,
leaving a maximum for T_1 of 64 μs.

2/2

For Interrupt 3
If T_1 = 0 there would be is 41 μs margin and T_1 could be called 1 time,
leaving a maximum for T_1 of 41 μs.

For Interrupt 3
If T_1 = 0 and T_{4+} = 13 μs there would be is 540 μs margin and T_1 could be called 8 times,
leaving a maximum for T_1 of 67.5 μs.

Conclusion: T_1 may be as long as 41 μs,
which is **10 μs longer than specified in case (a)**.

d.) If the T_{Pi} interrupts always happen at the minimum times between interrupts specified in the initial problem statement (meaning the interrupts are periodic and the minimum is also the maximum), then for case (a), what percentage of CPU time is available to the mainline routine, on the average?

1/1

As calculated in part (a), interrupts consumes 69.3 percent of the CPU time.
That leaves **30.7 percent of the CPU's time for the main code**.

3.19 **Critical Regions**. It is of paramount importance to be able to disable interrupts to protect critical regions in our code. Consider, for example, the common technique for changing a bit on an output port while leaving the other bits unchanged. We can read the port (even though it is an output port) into an accumulator. Then we can force the selected bit (e.g., bit 5) to 1 with an OR instruction (leaving the other bits unchanged). Finally, we can write the result back out to the port.

a.) Assuming that there is an interrupt service routine which changes bit 3 on this same port when it is called, why should interrupts be disabled for the above operation which is intended to change bit five? To answer this, describe a scenario which leads to a malfunction.

3/3 Suppose that between the time the mainline code reads the port (assume bits 3 and 5 are not set) and the time when the mainline code writes the information back, the interrupt occurs. The ISR will read the port (bits 3 and 5 are not set) and then change (say set) bit 3 as it should and then return to the mainline code. The mainline code (still working with the copy of the word in which bits 3 and 5 are not set) will then continue to finish what it started. It will set bit 5 (bit 3 is not set in this operation) and write the word back to the port. As intended, bit 5 gets set, but unintentionally, the work of the ISR was overwritten and bit 3 got cleared.

b.) How often is such an error likely to recur if bit 3 is changed once per millisecond, with the three instructions taking 1 μs each, and if bit 5 is changed once every 10 ms or so? Assume that the two events are not synchronized to each other, but that at least one of them occurs in response to an unsynchronized, external event.

The mainline code changes bit 5 every 10 ms.
The ISR changes bit 3 every 1 ms.
The read, OR, write sequence of operations each take 1 μs.

Every 10 ms or 10 000 μs there is a 2 μs long critical region (from the start of the read to the start of the write instruction). This means that each randomly occurring interrupt has a 2/10000 = 0.0002 chance of causing an error, or a 0.9998 chance of success.

3/3 **Over a 1000 second interval there will be 10000 critical regions. Each one of these has a 0.0002 chance of causing trouble. Thus, there will be $10000 \times 0.0002 = 2$ errors. Thus there will be on average 500 seconds between errors. That is, on average, 8 minutes and 20 seconds between errors.**

c.) Is there any benefit to be had in an instruction which sets (or clears or toggles) selected bits of a port *directly*? Explain. (Not all microcontrollers have such an instruction, but some do, including both the Motorola 68HC11 and the Intel 8096.)

1/1 **Yes, since a single instruction cannot be interrupted, it will eliminate critical regions.**

Problems 3.17 and 3.19 are derived from nearly identical problems in Peatman, John B, *Design With Microcontrollers*, McGraw Hill, 1988.