**HW Test 9: Interrupts-General**

When the X-bit of the CCR is **[a]**, the non-maskable interrupt is enabled.

**Zero**

Which of the following are potential sources for maskable interrupts?

**Port H, Port J, PWM, SCI, SPI, Port P**

The \_\_\_\_\_ register is used to "boost" the priority of an interrupt source.

**HPRIO**

\_\_\_\_\_\_\_ interrupts can occur in three ways: an interrupt request on the \_\_\_\_\_\_\_ pin, an unimplemented software instruction trap, or a software interrupt instruction, \_\_\_\_\_\_\_.

**non-maskable, XIRQ, SWI**

An interrupt vector is \_\_\_\_\_\_\_ bits in the 9S12.

**16**

\_\_\_\_\_\_ interrupts can be disabled and enabled through software.

**Maskable**

Interrupts can be divided into two types: \_\_\_\_ which can happen at any time, and \_\_\_\_ which occur at predetermined intervals.

**Asynchronous | Synchronous**

Write a single C instruction using a bitwise operator that toggles pin 1 on Port H without affecting the value of any other pin.

**PTH = PTH ^ 0x02;**

To have the processor run runit() when timer channel five produces an interrupt we would run the following instruction: **SETVECTOR([a], [b]);**

**[a] = 0xFFE4 | [b] = runit**

The **[a]** bit in the CCR controls the non-maskable interrupt of the 9S12.

**x**

Write a C instruction to reset the interrupt flag for pin 1 of Port H.

**PIFH = PIFH | 0x02;**

When the X-bit of the CCR is , the non-maskable interrupt is disabled.

**1**

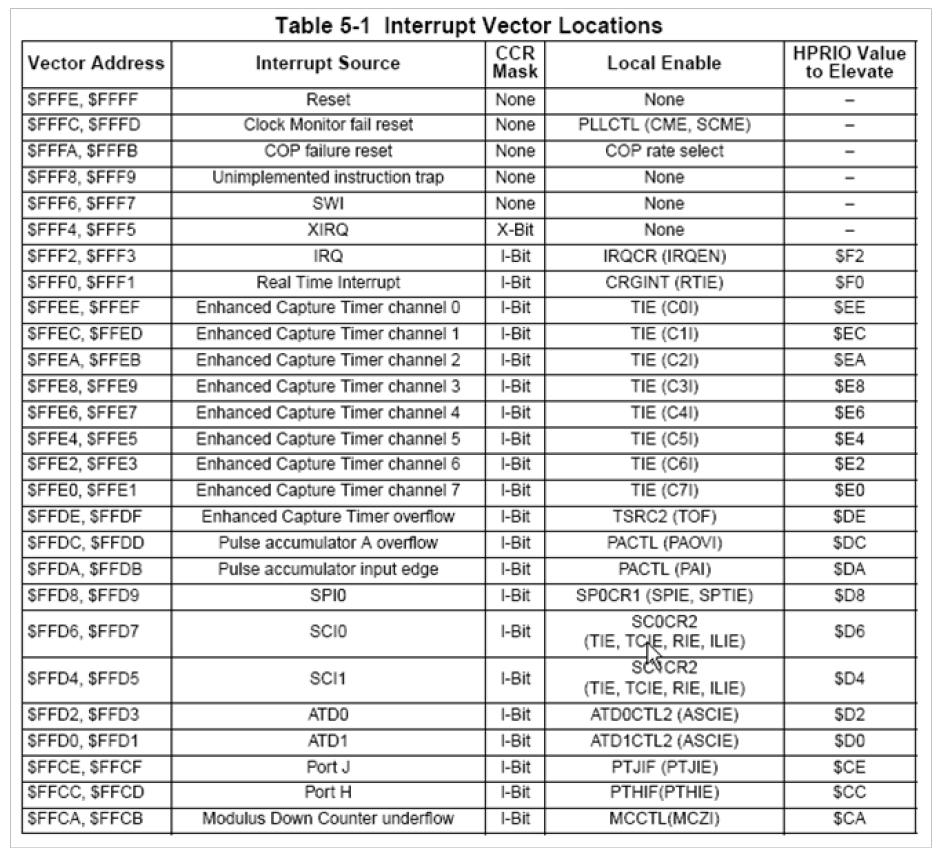
If programmed to, a[n] **[a]** reset can occur when a processor has entered an infinite loop.

**COP/Computer operating properly/Watchdog**

An interrupt vector is [a] bits in the 9S12.

**16**

Which of the following are potential sources of maskable interrupts?



**PWM, Port J, SCI, Port H, SPI, Port P**

What does SETVECTOR(0xFFCC, my\_isr) do?

**Tells processor to put address of my\_isr at address 0xFFCC**

A \_\_\_\_\_\_\_ occurs when a low-active reset pin is activated at startup to guarantee a known initial state.

**Power-On Reset**

A \_\_\_\_\_ is an exception with a “one-way ticket.”

**Reset**

To have the processor run doit() when pin 1 of Port H goes low, we would run the following instruction:

SETVECTOR([a], [b]);

**SETVECTOR(0xFFCC, doit);**

Write a single C instruction to reset only the interrupt flags on Port H which are high.

**PIFH = 0xFF;**

To have the processor run doatod() when timer the A/D 1 produces an interrupt we would run the following instruction: SETVECTOR([a], [b]);

**SETVECTOR(0x,FFCC, doatod)**

To have the processor boost the priority of an interrupt on pin 5 of Port H, we would put a value of [a] in the [b] register.

**a = 0xCC b = HPRIO**

The [a] bit of the CCR controls the maskable interrupts of the 9S12

**I**

There are two types of interrupts which are determined by their timing. [a] interrupts can be used to switch between processes at pre-determined intervals.

**Synchronous**

There are two types of interrupts which are determined by their timing. An example of a/an [a] interrupt is a timer used to write to a display at specified periodic intervals.

**Synchronous**

There are two types of exceptions: [a] e.g. and external interrupt, and [b], e.g. a divide by zero.

**Hardware, Software**

See above Table (table in previous question). To have the processor boost the priority of an interrupt on timer 7, we would put a value a [a] in the [b] register.

**0xE0, HPRIO**

A[n] [a] reset occurs when the processor detects the system clock is not within a valid frequency range.

**Clock Monitor**

[a] exceptions can be caused by illegal or erroneous instructions.

**Software**

When the X-bit of the CCR is [a], the non-maskable interrupt is enabled.

**0**

To have the processor run my\_isr() when pin 3 of Port H goes low, we would run the following instruction: SETVECTOR([a], [b]);

**SETVECTOR(0xFFCC, my\_isr);**

If programmed to, a[n] [a] reset can occur when a processor has entered a brown-out (low-voltage) condition.

**COP/Computer operating property/ Watchdog**

[a] interrupts can only be disabled through internal processor hardware.

**Non-maskable**

An [a] is a break in normal program flow and comes in two types: interrupts and resets.

**Exception**

The X-bit of the CCR cannot be set to [a] through software.

**1**

When the I-bit of the CCR is [a], all maskable interrupts are potentially enabled.

**0**

A[n] reset occurs when a low-active pin is activated by a user pressing a switch to force the processor to reset.

**External**

When the I-bit of the CCR is [a], all maskable interrupts are disabled.

**1**

Write a single C instruction using a bitwise operator that toggles pin 1 on Port H without affecting the value of any other pin.

**PTH = =PTH ^ 0x02;**

Write a single C instruction to reset the interrupt flags for every pin on Port H.

**PIFH = 0xFF;**

**HW Test 10: RTI General**

If the ECLK frequency is 7.6 MHz, what is the Core Clock frequency?

Core\_Clock\_Freq = 2\*ECLK\_Freq

**15.2**

If the Core Clock frequency is 11.6 MHz, what is the ECLK frequency?

ECLK\_Freq = Core\_Clock\_Freq/2

**5.8**

What bit in the 9S12 turns on or off the RTI?

**RTIE**

What bit in the 9S12 tells whether or not the RTI has expired?

**RTIF**

What clock is the RTI frequency directly based off?

**Oscillator Clock**

Given the following register settings in the 9S12

**RTICTL = 0x00**

**CRGINT = 0x92**

**CRGFLG = 0x8A**

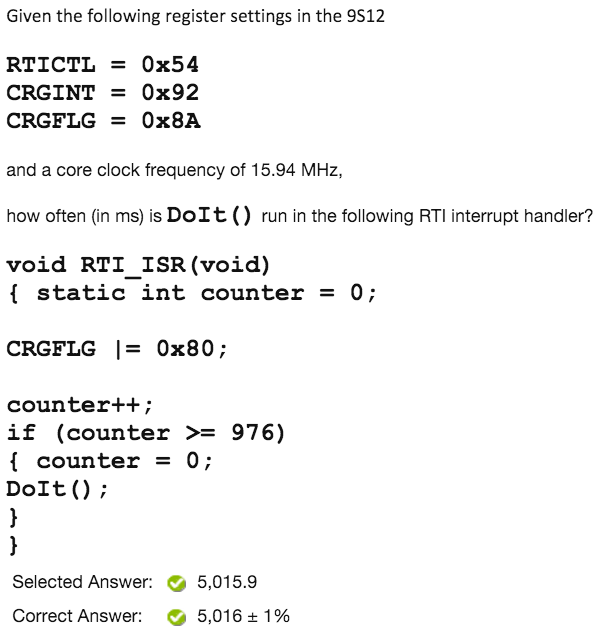
and a Core clock frequency of 10 MHz

what is the RTI frequency in Hz?

RTICTL determines interrupt frequency, so at 0x00, interrupts are disabled

**0**

**HW Test 10: RTI - What is Counter Value**



**RTI Divisor = 1024\*(RTICTL[lower in decimal]+1)\*2^(RTICTL[upper in decimal]-1)**

**RTI Rate = CCF/RTI Divisor**

**Seconds = Counter / RTI Rate**

|  |
| --- |
|  |

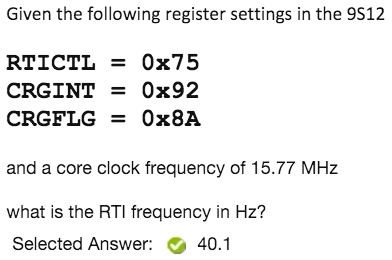
**RTI Divisor = 1024\*(RTICTL[lower in decimal]+1)\*2^(RTICTL[upper in decimal]-1)**

**RTI Rate = CCF/RTI Divisor**

**Counter = Seconds \* RTI Rate**

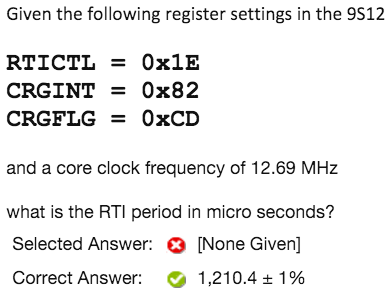
**(Always round to nearest integer)**

**HW Test 10: RTI-What is Frequency**



**RTI Divisor = 1024 \* (RTICTL(lower in decimal) +1) \* 2^(RTICTK(upper in decimal)-1)**

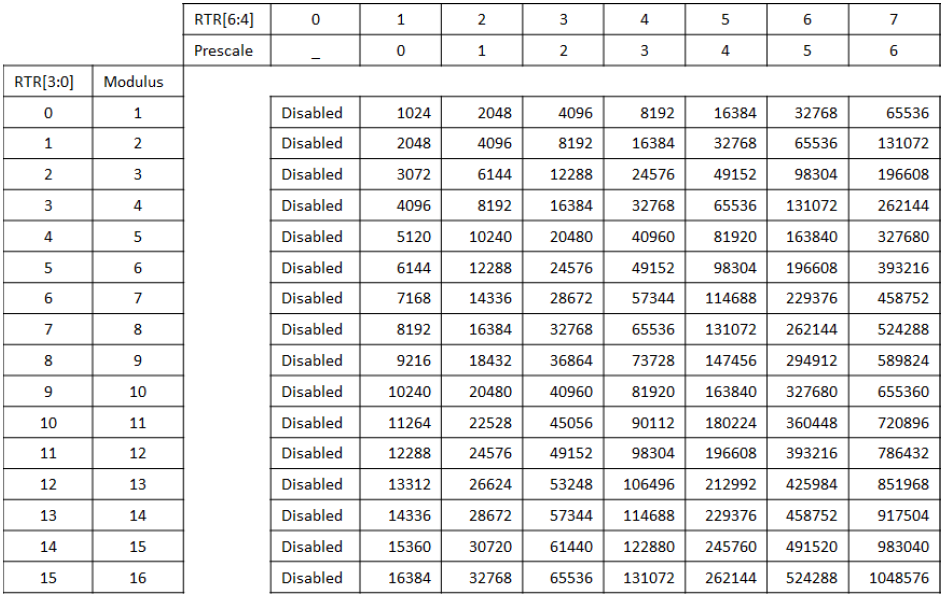
**RTI Frequency = Core Clock Frequency (MHz) / RTI Divisor**

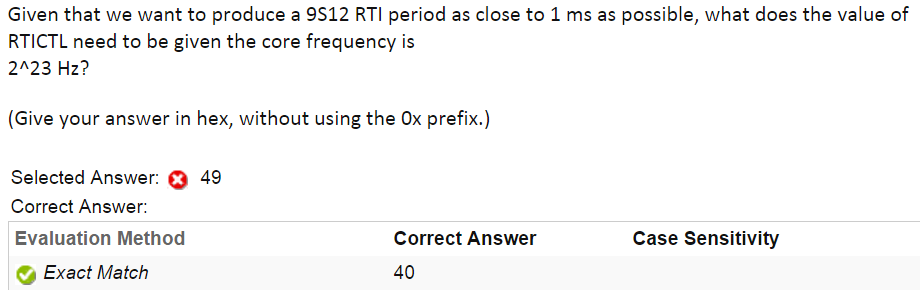


**RTI Divisor = 1024 \* (RTICTL[lower in decimal] + 1) \*2^(RTICTL[upper in decimal] - 1)**

**Period (ms) = RTI Divisor / Clock Freq (MHz)**

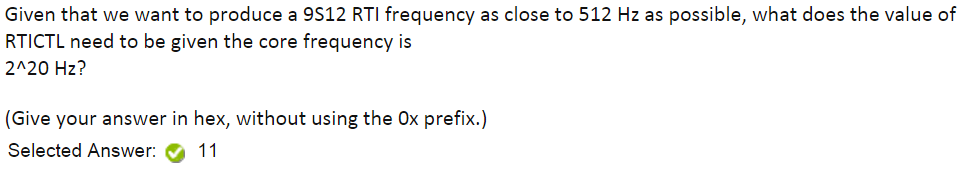
**HW Test 10: RTI-What is RTICTL**





**Table # = Core Frequency \* Seconds**

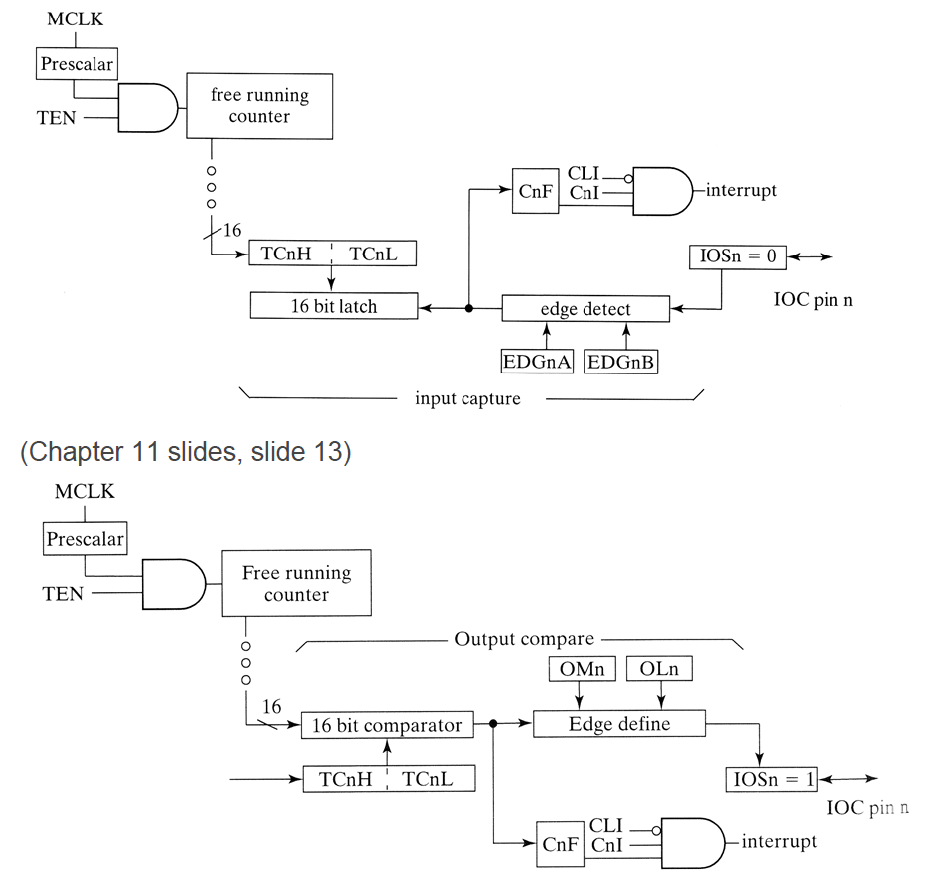
**Find upper and lower bits by plugging number into table**



**Table # = Core Frequency / Hz**

**Find upper and lower bits by plugging number into table**

**HW Test 11: Timers General**



Which Timer Channel can affect all other Timer Channels' behavior.

**Channel 7**

What 9S12 integrated functionality counts clock pulses until an input signal changes state?

**Input Capture**

What 9S12 integrated functionality works like a “stop watch?”

**Input Capture**

What 9S12 integrated functionality works like a "Alarm Clock?"

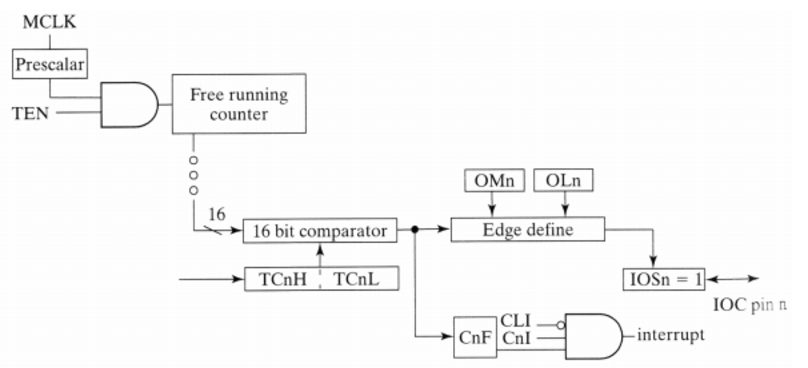
**Output Compare**

What does the following line of C instruct the compiler to do? Why/when is it necessary?

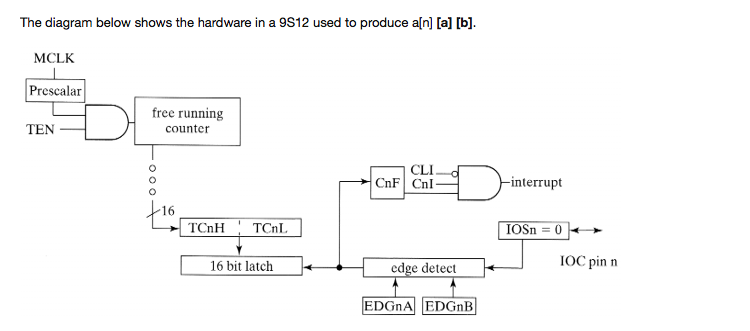
#define \_LP(x) \*(unsigned int \*)(x)

**It replaces \_LP(x) with \*(unsigned int \*)(x) where x is a variable name or constant. This statement is used with 16 bit operations.**

The diagram below shows the hardware in a 9S12 used to produce a[n] **[a]** **[b]**.

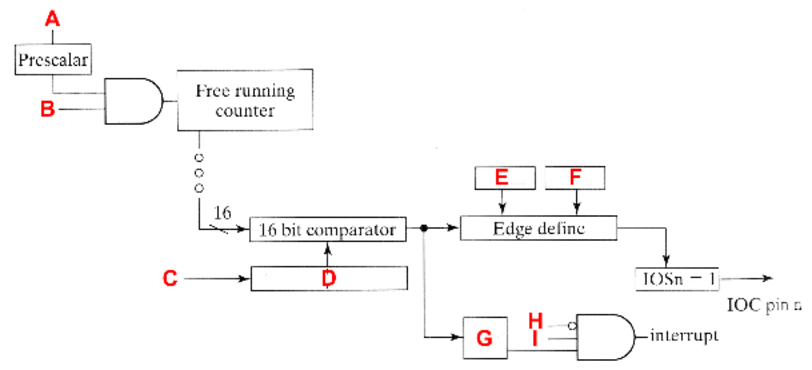


**Output Compare**



**Input Capture**

In the diagram below, what signal/bit corresponds to the letter **H**? **I**?



**(See Picture at Beginning)**

What signal values/transitions can the output compare system produce at a pin?

**Toggle, Low Value, High Value**

How many input capture channels are there on the 9S12?

**8**

How many Output Compare channels are there on the S12?

**8**

What signal values/transitions can input capture be initialized to detect?

**Rising Edge and Falling Edge**

When does the TOF bit get set?

**When the free-running counter goes from 0xFFFF to 0.**

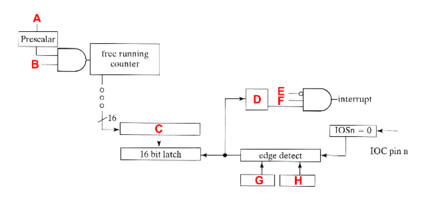
What is the lowest value possible in the TCNT register?

**0, zero, 0x0000**

What is the highest value possible in the TCNT register?

**0xFFFF**

In the diagram below, what signal/bit corresponds to the letter **B**? **G**?



**(SEE Picture at Beginning)**

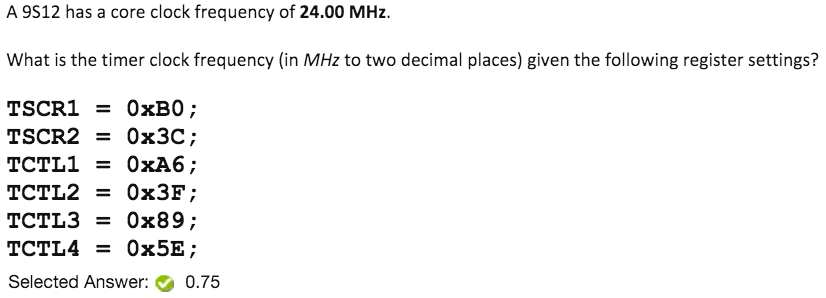
What signal values/transitions can the output compare system produce at a pin?

**Toggle, Low Value, High Value**

Which of the following registers are 16 bits?

**TC4, TC5, TCNT**

**HW Test 11: Frequency**

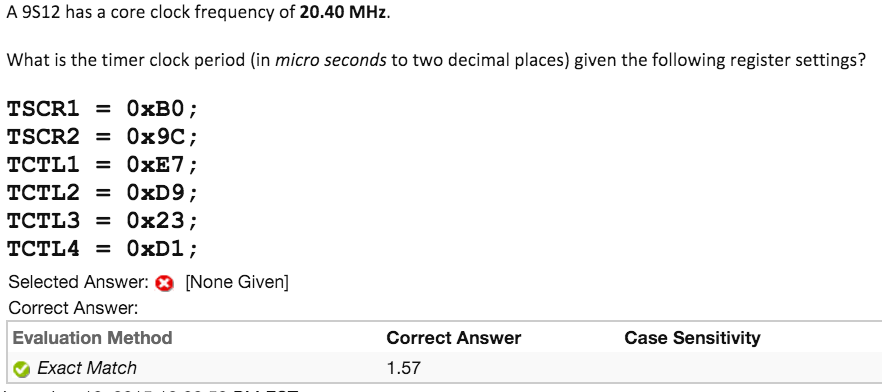


**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

**Bus Clock = Core Clock / 2**

**Frequency = Bus Clock / PR**

**HW Test 11: Period**



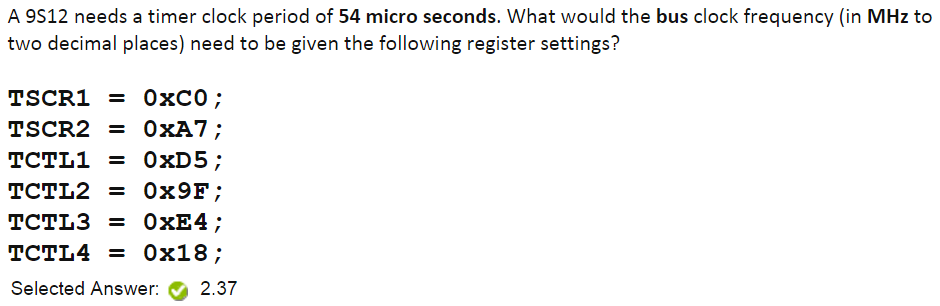
**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

**Bus Clock = Core Clock / 2**

**Frequency = Bus Clock / PR**

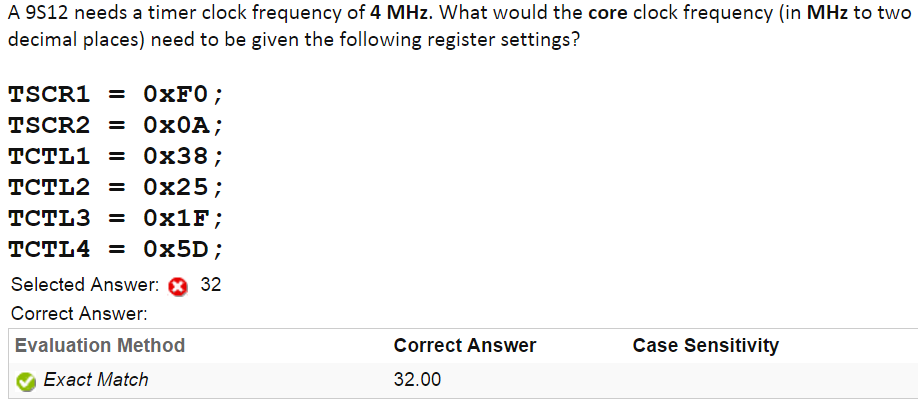
**Clock\_Period = 1/Frequency = PR / Bus Clock**

**HW Test 11: Clock for Period or Frequency**



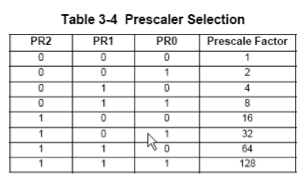
**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

**Bus Clock = PR / Clock\_Period**

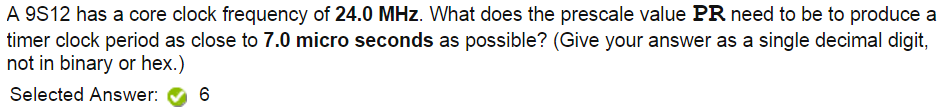


**CCF = Clock\_Freq\*2\*2^(TSCR2(first 3 bits of lower[binary]))**

**HW Test 11: Prescale**



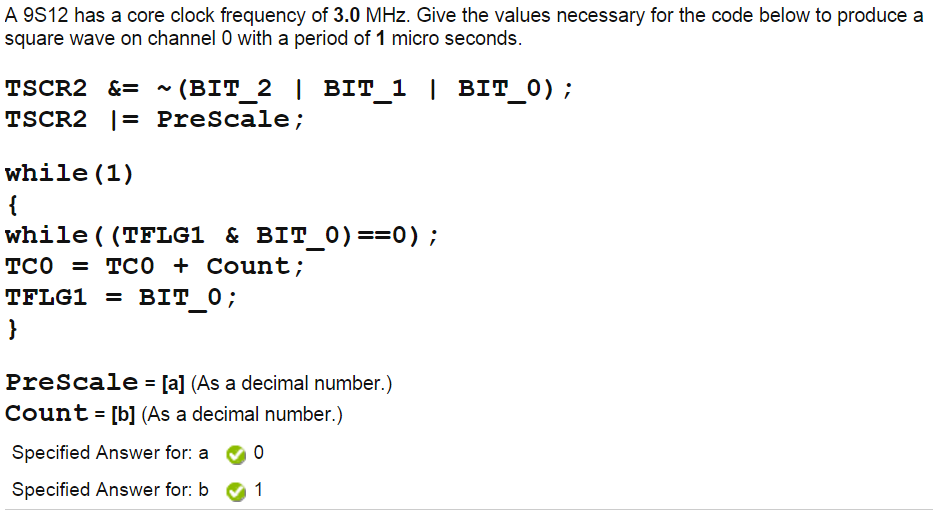
**Match with closest Prescale Factor**



**Bus Clock = Core Clock / 2**

**PR = Bus Clock\*Clock\_Period**

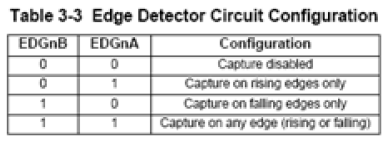
**HW Test 11: Prescale and Count**



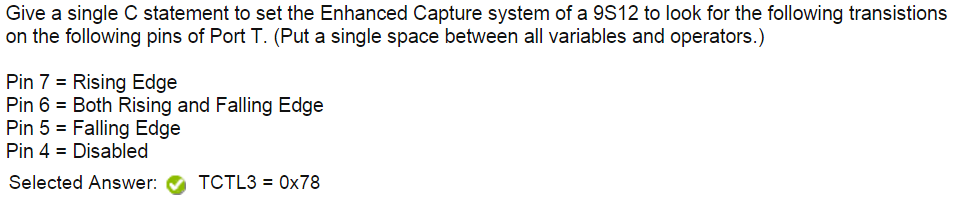
**(2^PR)(COUNT) = (BusClk\*Period)/2**

**\*pick value of PR to get whole # for Count**

**HW Test 11: Set Edge Detection**



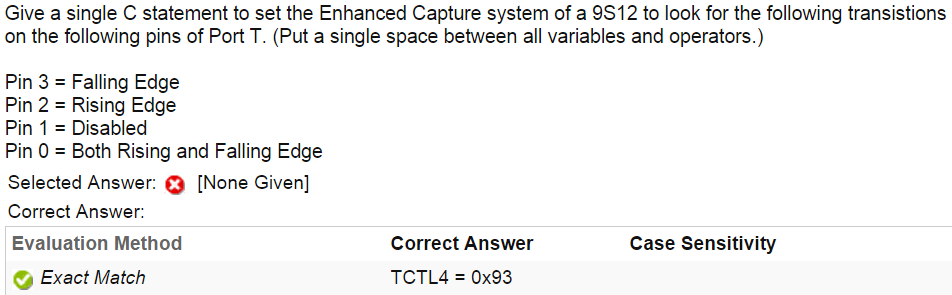
**\*TCTL3 [7:4] \*TCTL4[3:0]**



**Compare with table**

**Make each pin a two digit Binary Number using B and A**

**Convert to Hex**

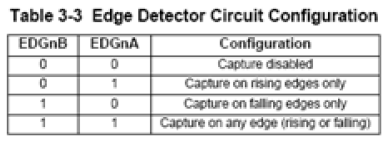


**Compare with table**

**Make each pin a two digit Binary Number using B and A**

**Convert to Hex**

**HW Test 11: What Edge Detection**



**Pin 7 - Bit 7 and 6 of TCTL3**

**Pin 6 - Bit 5 and 4 of TCTL3**

**Pin 5 - Bit 3 and 2 of TCTL3**

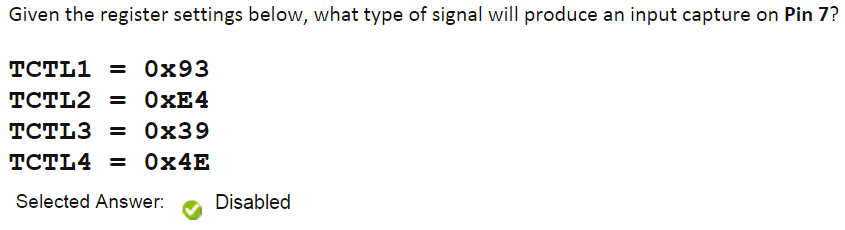
**Pin 4 - Bit 0 and 1 of TCTL3**

**Pin 3 - Bit 7 and 6 of TCTL4**

**Pin 2 - Bit 5 and 4 of TCTL4**

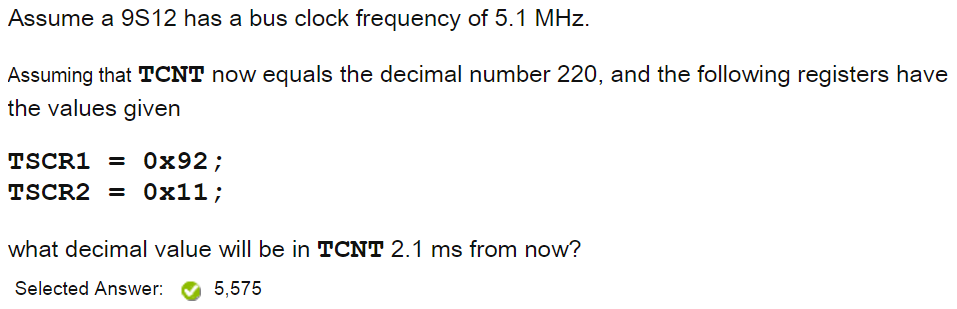
**Pin 1 - Bit 3 and 2 of TCTL4**

**Pin 0 - Bit 1 and 0 of TCTL4**



**Use Directions and Table above**

**HW Test 11: TCNT Calc**

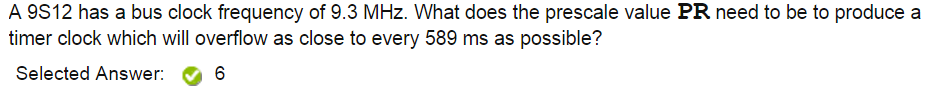


**Bus\_Clock = Core\_Clock / 2**

**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

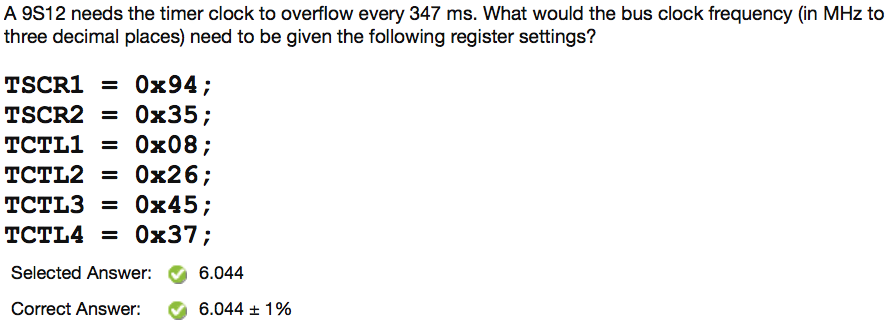
**TCNT = ((Bus\_Clock\*Time) / PR) + Inititial\_TCNT**

**HW Test 11: TCNT Max Count Calc**



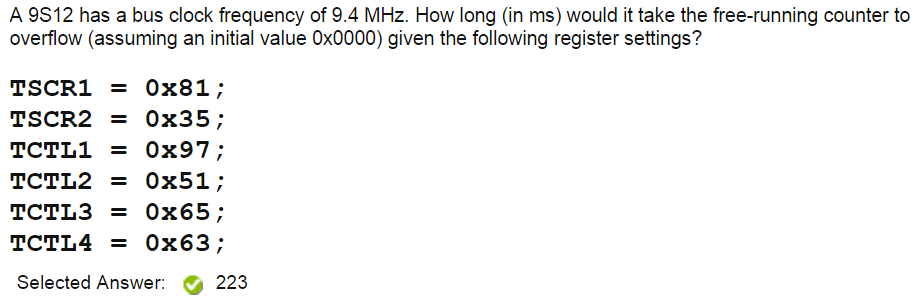
**Bus\_Clock = Core\_Clock / 2**

**PR = log2((Bus\_Freq \* Time)/65536 = 4.596**



**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

**Bus\_Clock = (65536\*PR)/Time(sec)**

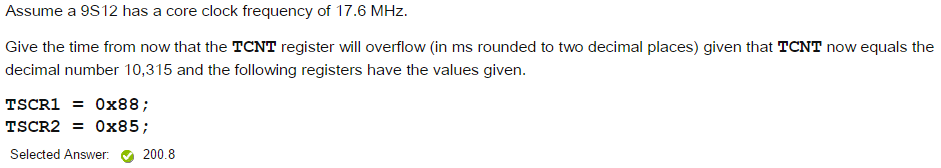


**Bus\_Clock = Core\_Clock / 2**

**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

**Time(sec) = (65536\*PR)/Bus\_Clock**

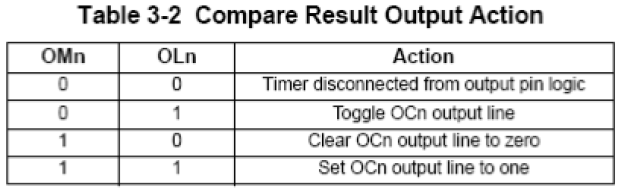
**HW Test 11: Overflow Calc**



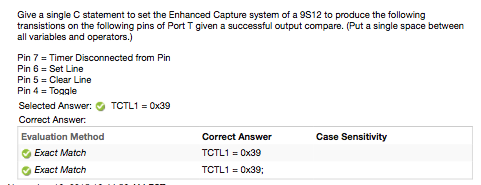
**PR = 2^(TSCR2(first 3 bits of lower[binary])**

**Time(sec) = (65536 - Initial\_Count) \* PR / Bus\_Clock**

**HW Test 11: Set Output Compare Levels**



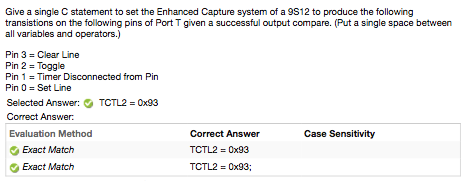
**\*TCTL1 [7:4] \*TCTL2 [3:0]**



**Compare with table**

**Make each pin a two digit Binary Number using B and A**

**Convert to Hex**

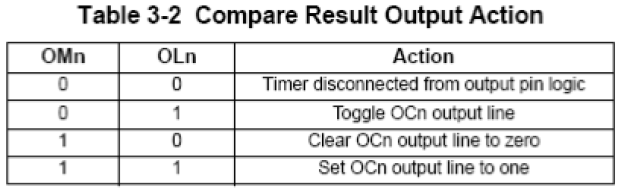


**Compare with table**

**Make each pin a two digit Binary Number using B and A**

**Convert to Hex**

**HW Test 11: What Output Compare Levels**



**Pin 7 - Bit 7 and 6 of TCTL1**

**Pin 6 - Bit 5 and 4 of TCTL1**

**Pin 5 - Bit 3 and 2 of TCTL1**

**Pin 4 - Bit 0 and 1 of TCTL1**

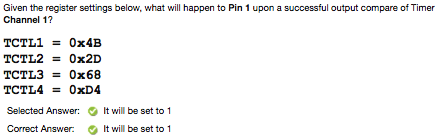
**Pin 3 - Bit 7 and 6 of TCTL2**

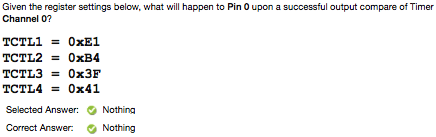
**Pin 2 - Bit 5 and 4 of TCTL2**

**Pin 1 - Bit 3 and 2 of TCTL2**

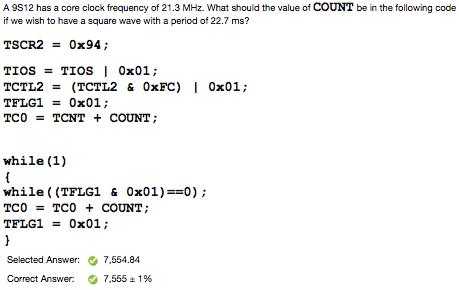
**Pin 0 - Bit 1 and 0 of TCTL2**

**Use Directions and Table Above**





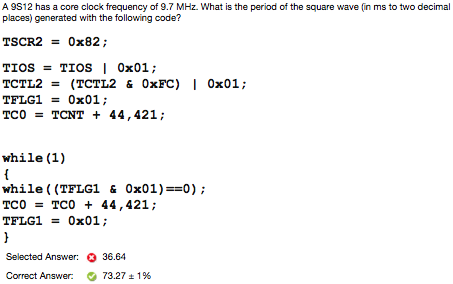
**HW Test 11: Square Wave Calc**



**Bus\_Clock = Core\_Clock / 2**

**PR = 2^(TSCR2(first 3 bits of lower[binary]) + 1)**

**Count = (Bus\_Clock \* Time(sec)) / PR**

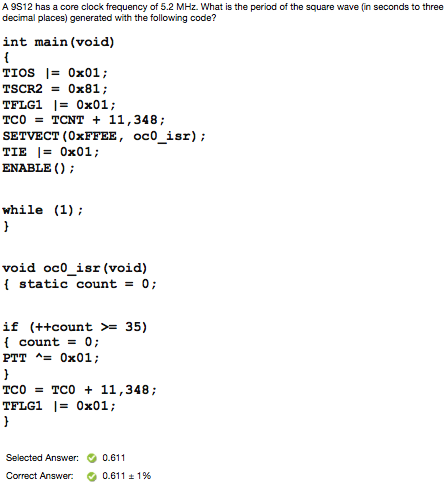


**Bus\_Clock = Core\_Clock / 2**

**PR = 2^(TSCR2(first 3 bits of lower[binary]) + 1)**

**Time(sec) = (Count \* PR) / Bus\_Clock**

**HW Test 11: Square Wave Calc with Overflow**

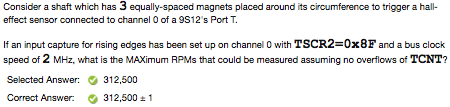


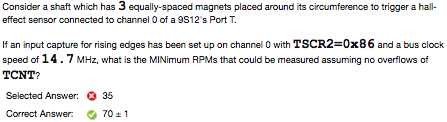
**Bus\_Clock = Core\_Clock / 2**

**PR = 2^(TSCR2(first 3 bits of lower[binary]) + 1)**

**Wave\_period = (Count \* Offset \* PR) / Bus\_Clock**

**HW Test 11: Application - Shaft - Max RPMs**





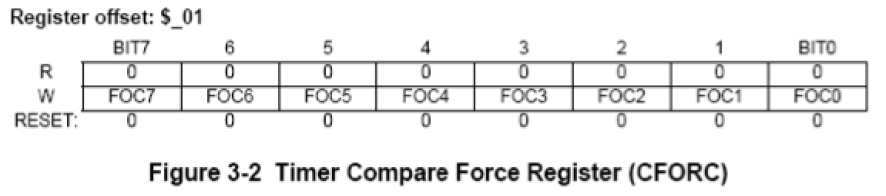
**PR = 2^(TSCR2(first 3 bits of lower[binary]))**

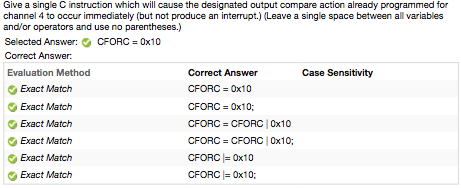
**If minimum, time = (65536 \* PR) / Bus\_Freq**

**If maximum, time = PR / Bus\_Freq**

**RPMs = 60 / (time \* #magnets)**

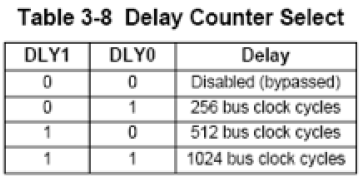
**HW Test 13: CFORC**

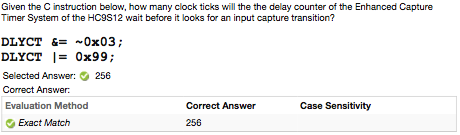




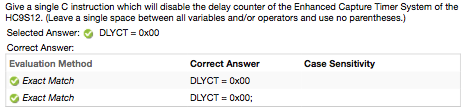
**Put a 1 where your channel is, everything else is 0. Convert to Hex**

**HW Test 13: Delay Count**





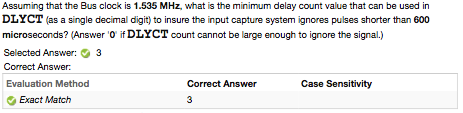
**Using Second Line of Code, Get bit 1 and bit 0 and match with table**



**Match number of cycles with table, that become bit 1 and bit 0**

**DLYCT =0x0#**

**HW Test 13: DLYCT Calc**



**Bus\_Clock = Core\_Clock / 2**

**Count\_Value = Bus\_Clock (Hz) \* Time(sec)**

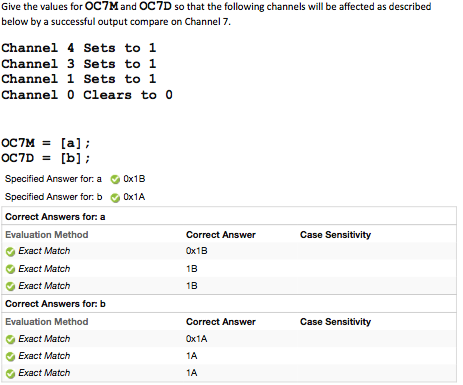
**Count\_Value < 256 => 1**

**Count\_Value < 512 => 2**

**Count\_Value < 1024 => 3**

**Count\_Value > 1024 => 0**

**HW Test 13: OC7M-OC7D**



**Write out bits - ???? ????**

**If bit not mentioned, replace with 0**

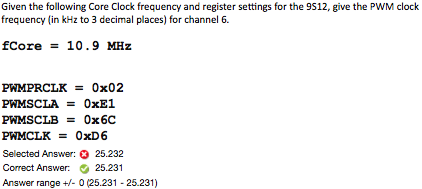
**If bit is sets to 1, replace with 1**

**For bits that clear to 0: OC7M will be 1, OC7D will be 0**



**If a bit is 1 in OC7M, then the bit from the OC7D overrides the bit in PTT, otherwise bit is same as PTT**

**HW Test 14: Find PWM Clock Freq**



**1. Bus\_Clock = Core\_Clock / 2**

**2. To determine whether it will be A/SA, or B/SB:**

**A/SA if Channel 0,1,4,5**

**B/SB if Channel 2,3,6,7**

**3. For PR:**

**If A/SA: PR = 2^(PWMPRCLK(bits[2:0]))**

**If B/SB: PR = 2^(PWMPRCLK(bits[6:4]))**

**4. Clock A/B = Bus\_Freq / PR**

**5. To determine whether S clock or not:**

**If the corresponding bit of your channel and PWMCLK is a 1, then it will be an S clock**

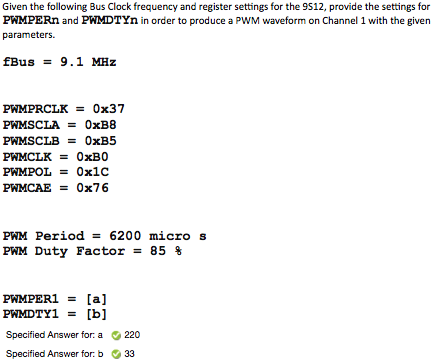
**Otherwise, the number you got in part 4 is your answer**

**6. If SA/SB clock**

**Sa/SB = (Clock A/B (Part 4))/ (2\*(Decimal value of either PWMSCLA/PWMSCLB))**

**\*DO NOT ROUND \* MAKE SURE ITS kHz**

**HW Test 14: Find PWMPER**



**1. Bus\_Clock = Core\_Clock / 2**

**2. To determine whether it will be A/SA, or B/SB:**

**A/SA if Channel 0,1,4,5**

**B/SB if Channel 2,3,6,7**

**3. For PR:**

**If A/SA: PR = 2^(PWMPRCLK(bits[2:0]))**

**If B/SB: PR = 2^(PWMPRCLK(bits[6:4]))**

**4. Clock A/B = Bus\_Clock / PR**

**5. To determine whether S clock or not:**

**If the corresponding bit of your channel in PWMCLK is a 1, then it will be an S clock**

**Otherwise, the number you got in part 4 is your answer (Skip to 7)**

**6. If SA/SB clock**

**Sa/SB = (Clock A/B (Part 4))/ (2\*(Decimal value of either PWMSCLA/PWMSCLB))**

**7. Period\_One\_Click = 1 / (Answer from part 4 or 6)**

**8. To Find PWMPER**

**If the corresponding bit of your channel in PWMCAE is a 0**

**PWMPER = (PWM Period(given)) / Period\_One\_Click**

**If the corresponding bit of your channel in PWMCAE is a 1**

**PWMPER = (PWM Period(given)) / (2 \* Period\_One\_Click)**

**9. To Find PWMDTY**

**If the corresponding bit of your channel in PWMPOL is a 0**

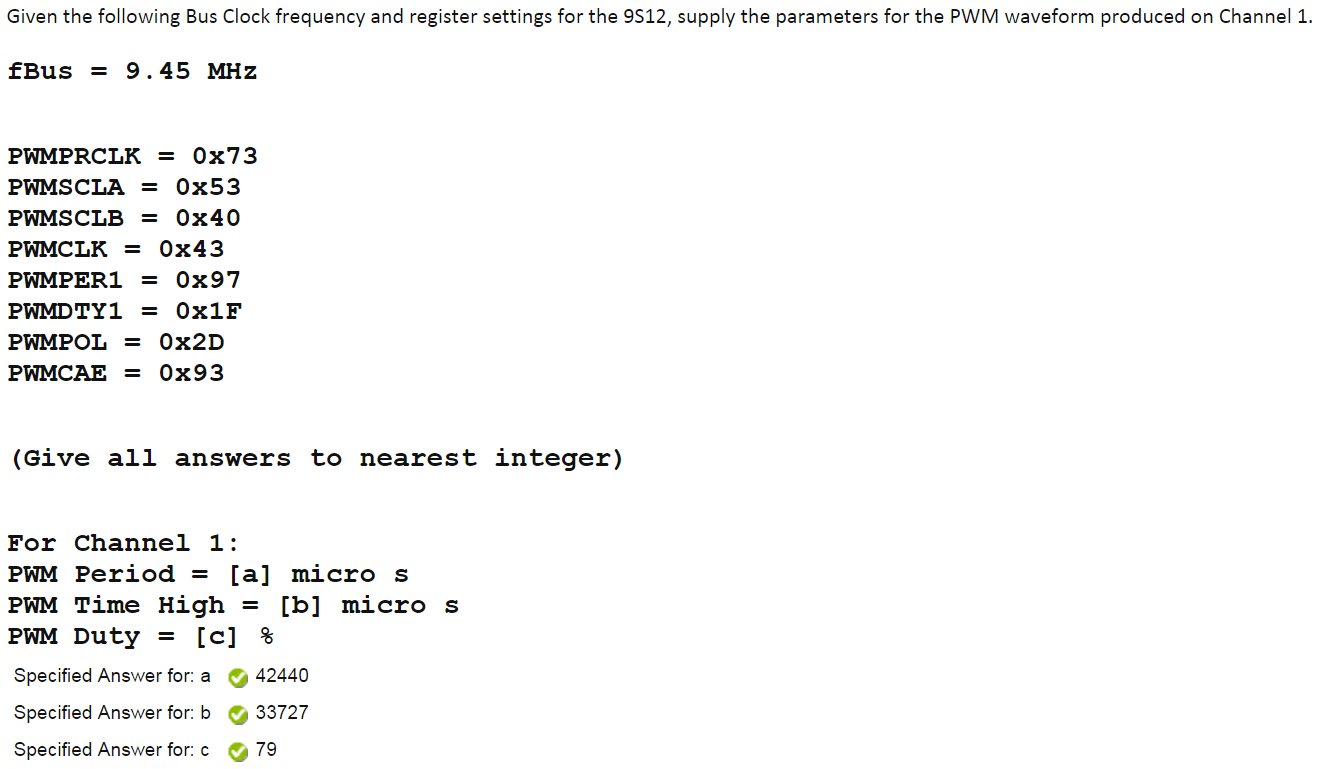
**PWMDTY = PWMPER \* (1-(DutyFactor(in %)/100))**

**If the corresponding bit of your channel in PWMPOL is a 1**

**PWMDTY = DutyFactor(in %) \* (PWMPER / 100)**

**\*ROUND TO NEAREST WHOLE NUMBER**

**HW Test 14: Find PWM PERIOD**



**1. Bus\_Clock = Core\_Clock / 2**

**2. To determine whether it will be A/SA, or B/SB:**

**A/SA if Channel 0,1,4,5**

**B/SB if Channel 2,3,6,7**

**3. For PR:**

**If A/SA: PR = 2^(PWMPRCLK(bits[2:0]))**

**If B/SB: PR = 2^(PWMPRCLK(bits[6:4]))**

**4. Clock A/B = Bus\_Clock / PR**

**5. To determine whether S clock or not:**

**If the corresponding bit of your channel in PWMCLK is a 1, then it will be an S clock**

**Otherwise, the number you got in part 4 is your answer (Skip to 7)**

**6. If SA/SB clock**

**Sa/SB = (Clock A/B (Part 4))/ (2\*(Decimal value of either PWMSCLA/PWMSCLB))**

**7. Period\_One\_Click = 1 / (Answer from part 4 or 6)**

**8. To Find PWM Period**

**If the corresponding bit of your channel in PWMCAE is a 0**

**PWM Period = PWMPER(Decimal) \* Period\_One\_Click**

**If the corresponding bit of your channel in PWMCAE is a 1**

**PWM Period = 2 \* Period\_One\_Click \* PWMPER(Decimal)**

**9. To Find PWM Duty**

**If the corresponding bit of your channel in PWMPOL is a 0**

**PWM Duty(%) = 100 \* (PWMPER(Dec) - PWMDTY(Dec)) / PWMPER(Dec)**

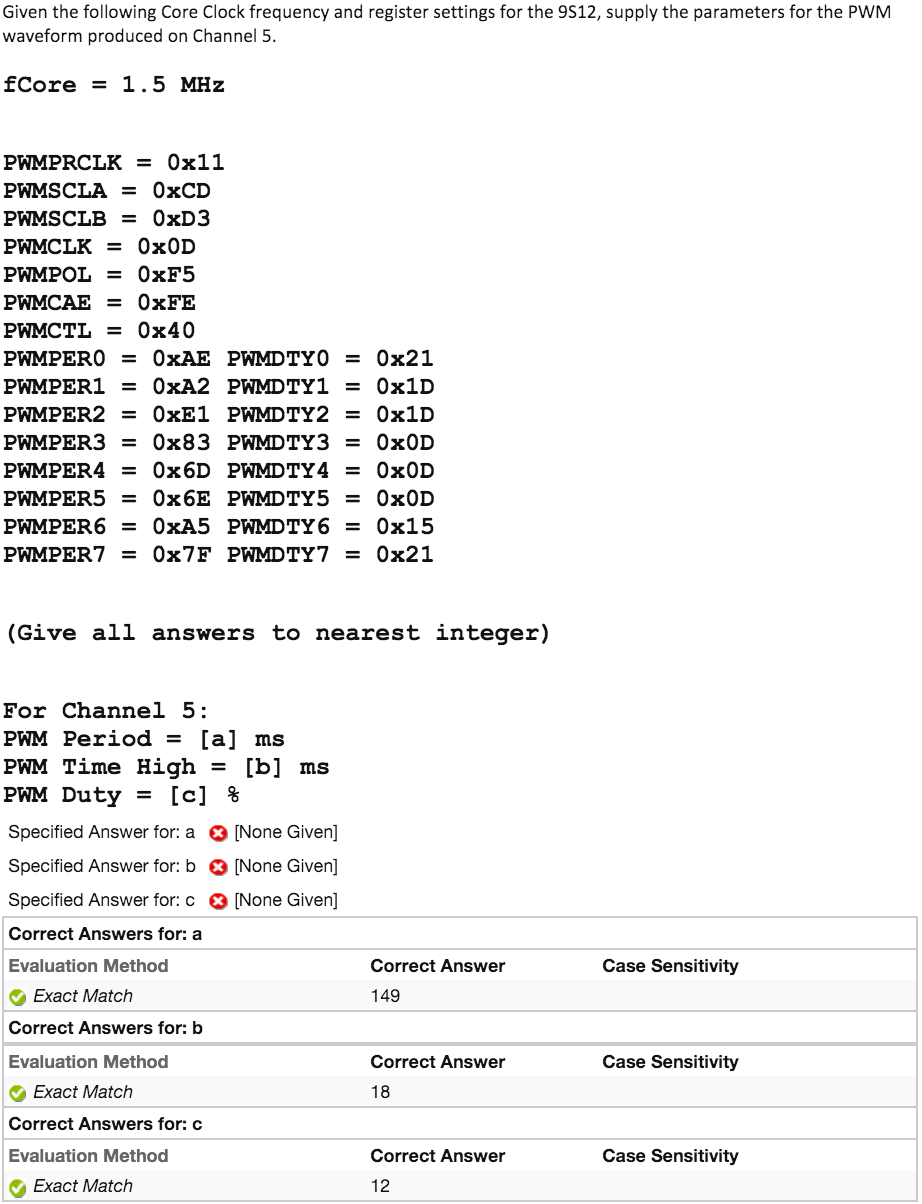
**If the corresponding bit of your channel in PWMPOL is a 1**

**PWM Duty(%) = 100 \* (PWMDTY(Dec) / PWMPER(Dec))**

**10. To Find PWM Time High**

**PWM Time High = PWM Period \* PWM Duty**

**HW Test 14: Find PWM-CON**



**HW Test 14: Find PWMCTRL**

Give a single C instruction which will stop the PWM input clock to the pre-scaler to save power in the 9S12 WAIT mode. (Leave exactly one space between each variable/symbol.)

**PWMCTL = 0x08**

Give a single C instruction which will stop all PWM counter in "freeze mode."

**PWMCTL = 0x04**

Give a single C instruction which will create a 16-bit PWM channel on pin 1 of Port P.

**PWMCTL = 0x10**

Give a single C instruction which will create a 16-bit PWM channel on pin 3 of Port P. (Leave exactly one space between each variable/symbol.)

**PWMCTL = 0x20**

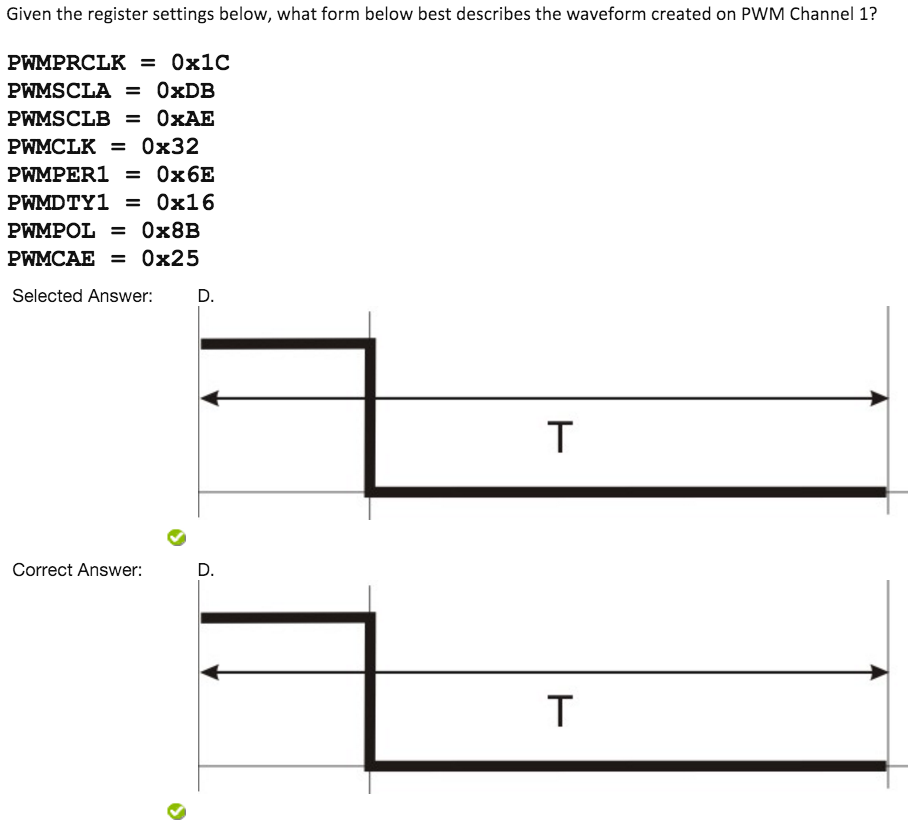
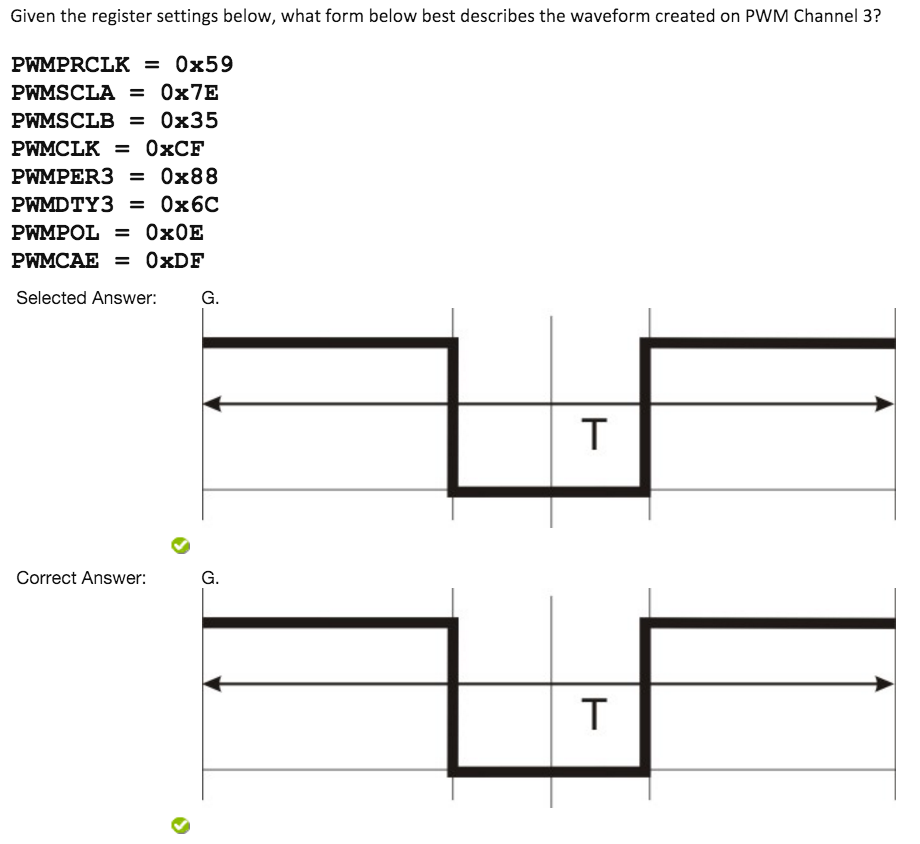
Give a single C instruction which will create a 16-bit PWM channel on pin 5 of Port P.

**PWMCTL = 0x40**

Give a single C instruction which will create a 16-bit PWM channel on pin 7 of Port P.

**PWMCTL = 0x80**

**HW Test 14: What PWM Form**



**PWMPOL to see if HIGH or LOW**

**LOW if corresponding Pin is 0**

**HIGH if corresponding Pin is 1**

**PWMCAE to see alignment**

**Left Aligned if corresponding Pin is 0**

**Center Aligned if corresponding Pin is 1**

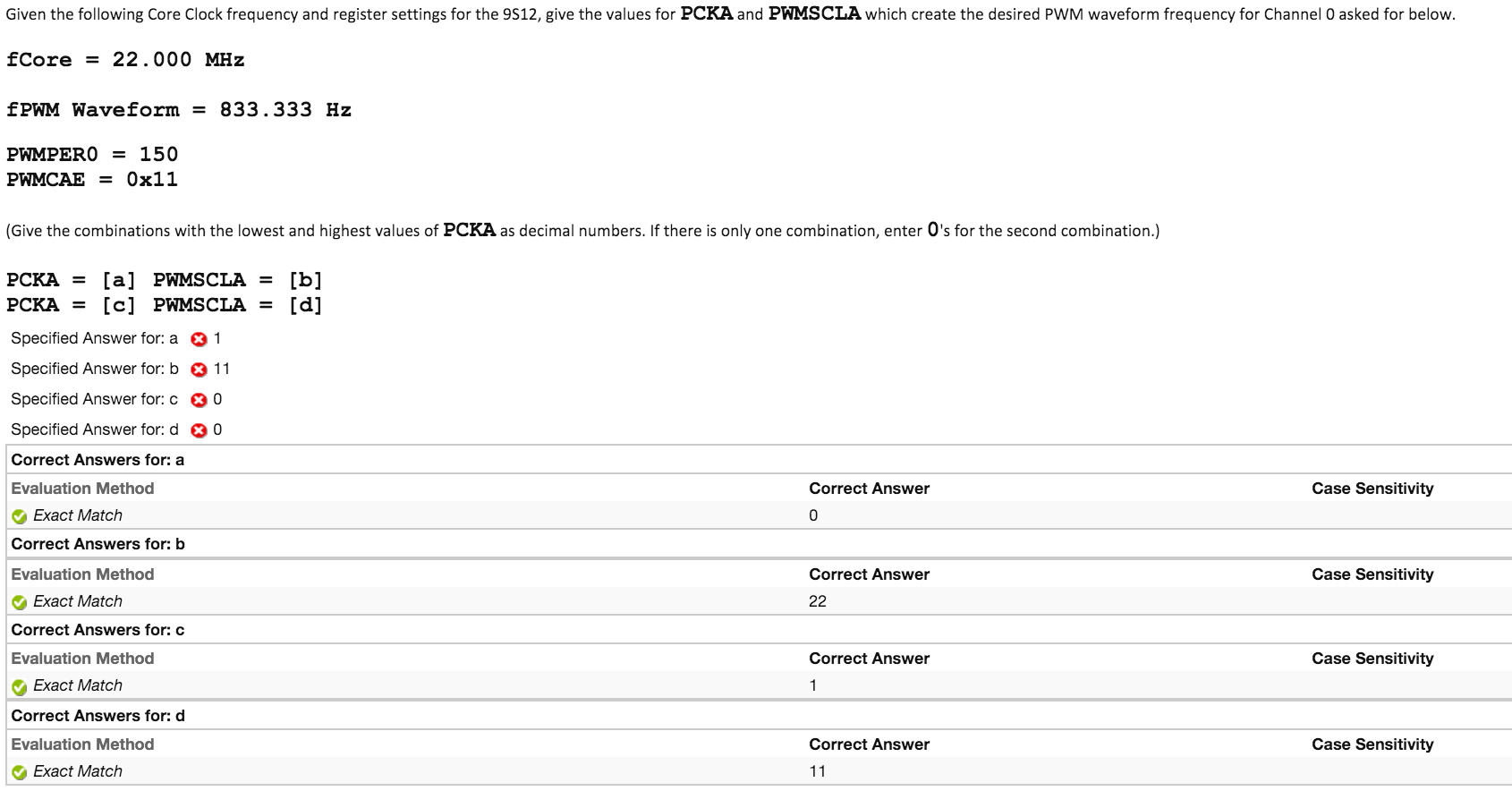
**PWMPER in Decimal is total Period**

**PWMDTY in Decimal is the time it is active in that period**

**If PWMDTY is close to PWMPER, then active for long time**

**So if active for long time, if starts low, find one that stays low the most. If starts high, find one that stays high most of the time**

**HW Test 14: What PWM Scale**



**Bus\_Freq = Core\_Freq / 2**

**(Bus\_Freq / fPWM(Hz)) / (2 \* PWMPER) = PWMSCLA \* (2^PCKA)**

**Plug in for left side of equations.**

**To solve for right side:**

**If Corresponding bit of PWMCAE is 1, divide entire left side of equation by 2**

**Make into whole number**

**Divide left side of equation by 2^(Highest value of something that will still give whole number) = PWMSCLA**

**The value that you raise (2^?) = PCKA**

**For the second set of numbers you need to find, SCLA can’t be above 255. If the whole number found previously is below 255: PCKA = 0; PWMSCLA = Whole number**

**Otherwise find the next value that will give a PWMSCLA below 255**

**(Alternate Way to solve in calculator)**

**Bus\_Freq = Core\_Freq / 2**

**Y = (Bus\_Freq / fPWM(Hz)) / (2 \* PWMPER)**

**If Corresponding bit of PWMCAE is 1, Y by 2**

**Calc: Solve(y=(2^?)\*x,x)**

**Starting at 0, raise the power up by 1. The first power that gives an x value below 255 is your first PCKA, and the x value is PWMSCLA.**

**Then continue to raise the power by one until your x value becomes a fraction, the power and x value before the fraction is your answer for the second set**