Definitions | Quiz | UT.6.01x Courseware | edX https://courses.edx.org/courses/UTAustinX/UT...
(https://www.edx.org)
UTAustinX: UT.6.01x Embedded Systems - Shape the World
KarenWest (/dashboard) ▼
Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)
Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)
Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)
Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)

DEFINITIONS (4/4 points)

Please match the following terms with the letter of their appropriate definitions.

nonintrusive D	**Answer:** D	A. Debugging by observing the inputs and outputs without looking inside.
black-box testing A	**Answer:** A	B. Debugging by controlling and observing the internal workings of a system.
white-box testing B	**Answer:** B	C. A minimally intrusive debugging instrument that stored strategic information in an array, and the contents of the array will be observed later.
dump C	**Answer:** C	D. A characteristic of a debugging instrument when the presence of the collection of information itself does not affect the parameters being measured.

Help

EXPLANATION

Black-box testing is simply observing the inputs and outputs without looking inside. Black-box testing has an important place in debugging a module for its functionality.

On the other hand, white-box testing allows you to control and observe the internal workings of a system.

Nonintrusiveness is a characteristic of a debugging instrument when the presence of the collection of information itself does not affect the parameters being measured. Nonintrusiveness is the characteristic or quality of a debugger that allows the software/hardware system to operate normally as if the debugger did not exist. Intrusiveness is used as a measure of the degree of perturbation caused in program performance by an instrument. For example, a print statement added to your source code and single-stepping are very intrusive because they significantly affect the real-time interaction of the hardware and software. When a program interacts with real-time events, the performance is significantly altered. On the other hand, an instrument with outputs strategic information on LEDs (that requires just 1 μs to execute) is much less intrusive. A logic analyzer that passively monitors the address and data by is completely nonintrusive. An in-circuit emulator is also nonintrusive because the software input/output relationships will be the same with and without the debugging tool.

A dump is minimally instrusive because it can execute in less than a μs.

Check Hide Answer(s)

SYSTICK COUNTER (1/1 point)

With the SysTick counter, what event causes the COUNT flag to be set?

- Whenever the counter is zero, and during the next clock the counter is reloaded from the RELOAD register.
- Whenever the system bus clock occurs.
- Whenever the software reads from the counter.
- [✓ Whenever the counter decrements from 1 to 0.] ✓
- Whenever an external interrupt event occurs.

EXPLANATION

The **Count** flag is set when the counter goes from 1 to 0. After 0 it is reloaded with the **RELOAD** value. So, if the bus clock is 12.5ns, the **Count** flag is set every (**RELOAD**+1)*12.5ns.

Check Hide Answer(s)

Help

SYSTICK COUNTER (1/1 point)

Which best describes SysTick?

- [✓ A 24-bit down counter decremented each bus cycle.] ✓
- A 24-bit up counter incremented each bus cycle.
- A 24-bit down counter decremented whenever an external event occurs.
- A 24-bit down counter decremented by executing software.
- A 32-bit down counter decremented each bus cycle.

EXPLANATION

Although all registers on the TM4C123 are 32 bits wide, SysTick implements only 24 of these bits. Many counters have prescales that you can set to adjust the count rate, but SysTick does not have a prescale; it counts down every bus cycle.

Check Hide Answer(s)

ARRAY INITIALIZATION (1/1 point)

What is the difference between these two array declarations? Both definitions create an array of four elements that will be initialized to the specified values before main is executed.

```
unsigned long array[4] = {1,2,3,4};
const unsigned long array[4] = {1,2,3,4};
```

- There is no difference.
- The first array is placed on the stack and the second in memory.
- The first array is placed in ROM and the second in RAM.
- [✓ The first array is placed in RAM and the second in ROM.] ✓
- none of the above

EXPLANATION

More formally **const** means the software cannot change the value at run time. On an embedded system, **const** tells the compiler to put this data in ROM. Without the **const** the compiler puts the variables in RAM.

Check | Hide Answer(s)

Help

BLANK ADVANCED PROBLEM (1/1 point)

For this array, long **array[7]**; what is the valid range of index **i** when accessing **array[i]?**

- 1 to 7
- [✓ 0 to 6] ✓
- 1 to 7
- 0 to 24
- none of the above

EXPLANATION

In C, arrays have **zero index**, meaning the first element is at index 0. Since there are 7 elements, the index can be 0, 1, 2, 3, 4, 5, or 6.

Check | Hide Answer(s)

STRINGS (1/1 point)

With an ASCII string in C, how do we tell how many characters are in the string?

- All strings are fixed size and we know in advance what that size is.
- There is a size count included with the string.
- [✓ After the last character there is a null, or zero value.] ✓
- The end of all strings terminate with a carriage return.

EXPLANATION

The **sentinel** in C is 0 or null. The null is automatically added by the compiler. For example the string "1234567" will require 8 bytes of storage: the 7 characters and the null.

Check Hide Answer(s)

CLEARING AN ARRAY (1/1 point)

For this fixed-size array of 100 unsigned integers, **unsigned long array[100]**; which C code clears the array?

```
void Clear(void){
  for(int i=0; i<100 ; i++){
    array[i] = 0;
  }
}
```
✓

[✓
```
void Clear(void){ int i;
  for(i=0; i<100 ; i++){
    array[i] = 0;
  }
}
```
] ✓

```
void Clear(void){ int i;
  for(i=0; i<100 ; i++){
    array[0] = 0;
  }
}
```

```
void Clear(void){ int i;
  for(i=0; i<99 ; i++){
    array[i] = 0;
  }
}
```

EXPLANATION

The first program is valid in C++ and C99. In C, we define local variables only after an open brace.

The third program only modifies the first array entry.

The four program does not modify the last array entry.

Check Hide Answer(s)

Help