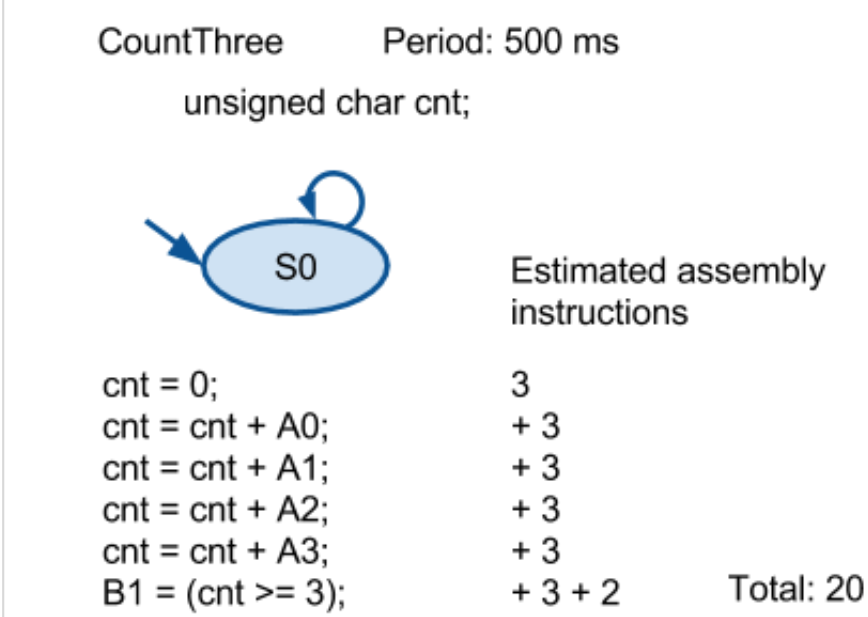
10.2 Utilization

10.3 Computing a task's worst-case execution time

For each tick of a task, the task's actions require some time to execute. When doing utilization analysis for C code executing on a microcontroller, a programmer can estimate that time by estimating the number of assembly instructions per C statement, and multiplying by the time the microcontroller takes to execute each assembly instruction.

Consider the following synchSM task named CountThree, with a period of 500 ms, that sets B1 to 1 if A0-A3 have three or more 1s:

Figure 10.3.1: CountThree synchSM.



Assume C statements translate to assembly instructions as follows:

- 3 assembly instructions for a simple C assignment statement, like cnt = 0 or B1 = cnt + A0.
- 2 assembly instructions for a C comparison, like cnt >= 3.

Given these assumptions, each tick of the above task requires 20 instructions, as shown in the figure.

Assume a particular microcontroller executes 800 assembly-level instructions per second, meaning 1 sec / 800 instr = 0.00125 sec/instr (this is a *very* slow microcontroller used for illustration only). Then the 20 instructions will require 20 instr * 0.00125 sec/instr = 0.025 sec = 25 ms. Utilization is thus 25 ms / 500 ms, or 5%.

Note that utilization analysis usually ignores the additional C instructions required to implement a task in C, such as the switch statement instructions in a tick function, or the scheduler code. For typical-sized tasks and typical-speed microcontrollers, the number of such "overhead" instructions is negligibly small.

The above analysis considered only simple assignment statements. If the expression on the right is more complex, the analysis might consider the time for an equivalent computation using simpler statements, where each statement has only a single arithmetic operation. For example, the statement `cnt = A0 + A1 + A2 + A3` might first be converted to the above simpler statements for analysis.

The above numbers are dependent on the particular microcontroller and also on the C compiler. Also, on some microcontrollers, certain arithmetic operations like multiply and divide might take longer than other operations like plus and minus. Microcontroller datasheets may provide details. Modern microcontrollers execute about 1 instruction per clock cycle, so a microcontroller with a 1 MHz clock executes 1 million instructions/sec, meaning 1 microsec per instruction, while a 1 GHz clock means 1 nanosec per instruction.

A thorough programmer might conduct experiments to determine the actual execution time of a series of statements on a particular microcontroller. Also, some analysis tools exist that will provide estimates.

Participation Activity 10.3.1: BasicCompET.

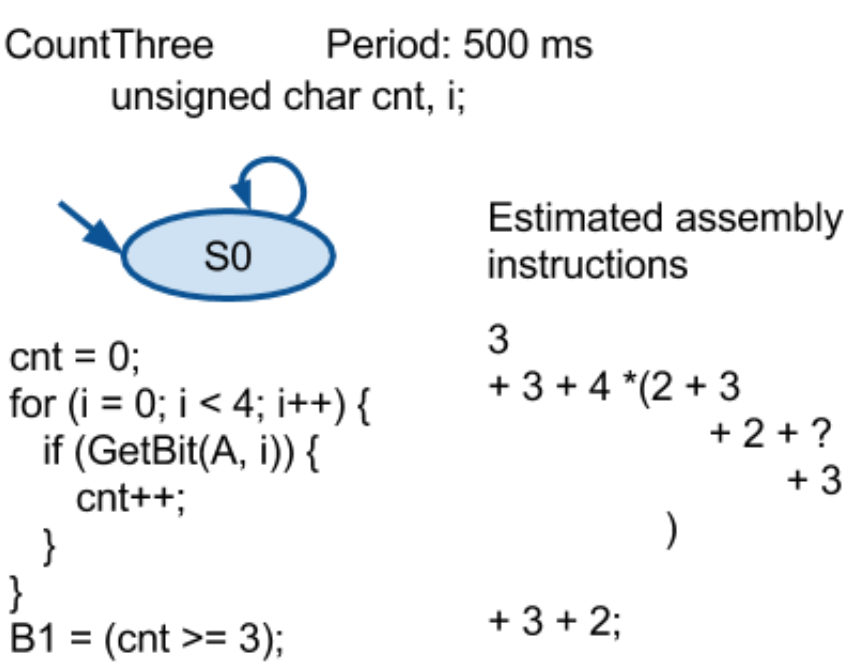
A synchSM has one state. Use the above numbers for translating C to assembly instructions. Assume each instruction requires 2 microsec to execute.

#	Question	Your answer
1	The state has actions B0 = 1, B1 = 0, B2 = 1, B3 = 0. How many microsec does one tick require? ✔ 24 3 + 3 + 3 + 3 instructions, 2 microsec/instr, so 12*2 is 24.	24 microsec
2	The state has the action B = X + Y + Z. How many microsec does one tick require? ✔ 12	12 microsec

2	For analysis, replace by tmp = X + Y, B = tmp + Z. 2 statements, 3 instrs each, 2 microsec/instr, so 2*3*2 is 12.	
3	If the synchSM has period 20 microsec and actions tmp = X + Y, B = tmp + Z, what is the utilization? ✔ 60 3 + 3 instrs * 2 microsec/instr yields 12 microsec. 12 microsec / 20 microsec is 60% utilization.	60 %
4	If a microcontroller has a 100 MHz clock and executes 1 instr per clock cycle, how long does 1 instr take to execute (in nanosec)? ✔ 10 1 / 100,000,000 is 0.000 000 01 , or 10 nanosec. An easy way to remember is: 1 GHz yields 1 nanosec, so a 10x slower clock yields 10x slower time per instruction. Likewise, remembering that 1 MHz yields 1 microsec can also help.	10 nanosec

A state's actions may include loops, function calls, branch statements, and more, as shown below.

Figure 10.3.2: CountThree synchSM with loops and branches.



Assume 3 instructions per simple C assignment statement, and 2 instructions per simple comparison expression. Assume the two instructions for the if expression comparison (even if just one item like A0) encompasses the if-else branching.

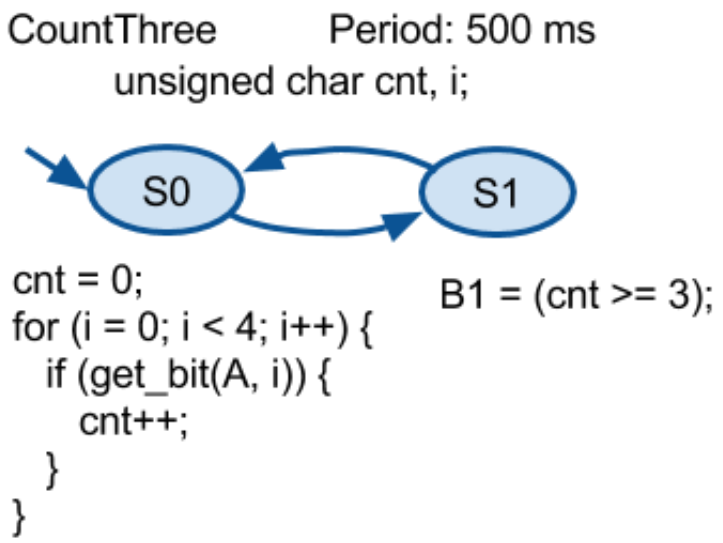
- For a *for* loop, the analysis should include the loop initialization (i=0: 3 instrs), plus the loop control instructions (i<4, and i++, so 2 + 3), and should also multiply the instructions-per-loop-iteration by the number of iterations. The above loop iterates 4 times. If the number of loop iterations is data dependent, an upper bound on the number of iterations should be used.
- For a function call, the analysis should determine the instructions executed within the function. Above, the number of instructions for the call to function GetBit (defined in an earlier section) is listed as "?". Examining the statements within the GetBit function itself, a programmer might estimate 10 instructions.
- For the *if* statement, the analysis should consider the *worst case*, which for this statement would mean the branch *is* taken and thus cnt++ is executed. In general, in the presence of branches (if-else statements), the analysis must consider the maximum number of instructions that might be executed for any values of the branch conditions.

Thus, the total worst-case number of assembly instructions that execute for S0's actions is 3+3+4*(2+3+2+10+3)+3+2 or 91 instrs. On a microcontroller that requires 0.00125 sec/instr, the worst case execution time for those instructions is 91 instr * 0.00125 sec/instr or 113.75 ms. For a period of 500 ms, the utilization is 113.75 ms / 500 ms, or 22.75%.

As should be clear from above, the ***worst-case execution time*** (or ***WCET***) of a synchSM task is determined as the time to execute the worst-case number of instructions for any possible tick of the synchSM. WCET is the value of concern regarding utilization and timer overrun.

For a task consisting of a multi-state synchSM, the analysis requires determining WCET for each state, and then using the max as the WCET for the synchSM. For example, consider the CountThree synchSM captured using two states:

Figure 10.3.3: CountThree synchSM with two states.



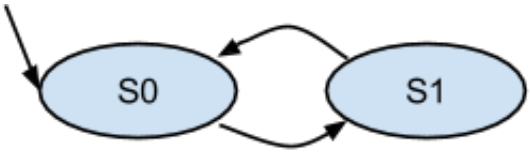
S0's actions translate to 3+3+4*(2+3+2+10+3) = 86 instrs, while S1's actions translate to 3+2 = 5 instrs. 86 instrs is larger than 5 instrs. On microcontroller M, the WCET is 86 instr * 0.00125 sec/instr = 107.5 ms. For a time window of 500 ms, the worst-case utilization of M is 107.5 ms / 500 ms, or 21.5%.

Any conditions and actions appearing on transitions should also be considered during analysis. One approach determines the worst-case number of instructions (due to conditions and actions) among all of a state's *incoming* transitions, and then adds that number to the state's actions. This approach makes sense when considering how a tick function, when translated to C, first executes a switch statement for transitions, and then executes a switch statement for the determined next state.

For the purpose of introduction, the above discussion uses very slow microcontrollers executing only a few hundred instructions per second. Typical microcontrollers will execute 10,000 to 10,000,000 instructions per second. Microcontroller utilization for the above example tasks would thus be well below 0.1%, as expected for such trivially simple tasks.

Participation Activity 10.3.2: synchSM WCET utilization.

Consider the following synchSM task.




```
tmp1 = A * 12;
if (A0) {
   tmp2 = tmp1 * tmp1;
}
tmp3 = tmp2 + 55;
```

```
for (i=0; i<100; ++i) {
   tmp4 = tmp3 * 2;
}
B = tmp4 % 7;
```

#	Question	Your answer
1	The WCET for S0 is involves approximately 3 (tmp1 =...) + 2 (if (A0)) + 3 (tmp3 = ...), or 8 instructions. ✔ The analysis must consider the worst case, which would mean the if-branch is taken and thus statement tmp2 = ... executes. So WCET involves 3 + 2 + *3* + 3 or 11 instructions.	True **False**
2	The WCET for S1 involves approximately 3 (i=0) + 2 (i <100) + 3 (++i) + 3 (tmp4 = ...) + 3 (B = ...), or 14 instructions. ✔ The for loop executes 100 times, so the equation should be 3 (i=0, only done before the loop) + *100* * (2 + 3 + 3) + 3, or 806. Loops tend to dominate execution time.	True **False**
3	The WCET is obtained as the minimum of the WCETs of S0 and S1. ✔ During any tick of the synchSM, one of those two states will execute. Thus, during any tick, the *worst*-case execution time is the *maximum* of the WCETs of S0 and S1, not the minimum.	True **False**

Exploring further: Wikipedia: WCET

10.4 Utilization for multiple tasks