Problem 1 [5, 10, 10 points] System Modeling

Consider the finite-state machine in the figure above that models switching of a traffic light. Initial state is “red”.

a. Write down a description of this FSM as a six-tuple describing the states, inputs, outputs, initial state, state transition, and output functions.

b. For input sequence, “tick, tick, tick..” write a trace of the FSM on including inputs, outputs, and states at each step. Is this the only trace possible on this input sequence? Why or Why not?

c. Now consider merging the “red” and “yellow” states into a single “stop” state. Transitions that pointed into and out of those states are now directed into or out of the new “stop” state. Other transitions and the inputs and outputs stay the same. The new “stop” state is the new initial state. Draw the resulting state machine. Is this state machine deterministic? Why or why not? For the input sequence in (b) above, write the trace (or traces) of the new machine.
Problem 2 [10 points] Models
Explain the main differences between the following graphical models: Petri Nets, Data Flow Graphs.

*Petri Nets capture both concurrency and choice in addition to sequencing. Data flow graphs, by contrast, capture only two of these: sequencing and concurrency.*

Problem 3 [3x5 points] Short Questions

1. Distinguish between separate compilation, independent compilation and multiprogramming.

*All three are different ways to compose multiple program units (bottom-up view) or divide a large program into modules (top-down view). This composition of program units can be static, i.e., orchestrated by the compiler or dynamic, i.e., orchestrated by the runtime system. Multiprogramming needs runtime support to run independently compiled program units. On the other hand, separate compilation is done within the context of a library. Once separate program is tested, it is added to the library where there may be a hierarchy of dependencies among modules. This allows libraries of precompiled program units and their reuse across programs.*

2. What is deterministic parallelism in synchronous languages?

*Deterministic parallelism refers to the property of programs in the synchronous languages to capture true parallel operations but guarantee the same results regardless of how parallel tasks are interleaved or assigned to processors. This simplifies any reasoning for correctness or timing for such programs. In otherwords, the language guarantees that the evaluation of parallel programs is deterministic. This is in contrast to concurrent programming.*
3. What is meant by resolution, precision and accuracy of a clock in embedded system?

Recall lecture by Thomas Schmid on clocking. A hardware clock component provides a clock signal that is used to update a clock counter. The clock counter is read by the embedded software.

A clock signal is a periodic signal of frequency $f_0$. The clock signal increments the clock counter every $1/f_0$ seconds. That is its resolution.

The smallest increment at which software can read the counter is its precision.

Finally, accuracy of a clock is a measure of how close is the clock counter to a time reference such as UTC.

[Note that localization can also be defined equivalently for its accuracy, precision and resolution.]

4. Explain the differences between termination- versus resumption-model of exception handling? Which would you use in case of synchronous exceptions? And In case of asynchronous exceptions?

These two models refer to actions after an exception has been handled by an appropriate routine. If the routine (exception handler) has cured the problem the invoker of the exception can continue its execution in the resumption model. This is useful in case of asynchronous exceptions since exceptions may have little to do with the current execution.

On the other hand, in case of termination model, the control is not returned to the original point of execution where the exception occurred. Instead the control goes to the calling block, procedure or process. This is useful in case of synchronous exceptions where further progress requires calling context to retry.