18-540
Distributed Embedded Systems
Test #1

September 27, 2000

Name (please print): _____________________________________________________

Instructions: DO NOT OPEN TEST UNTIL TOLD TO START
This test lasts from 12:30 PM to 2:20 PM

The test is composed of four problems (one of which with two sub-problems), adding up to 100 points overall. Attempt all problems and budget your time according to the problem’s difficulty. There should be more than enough time to complete the entire test. Show all work in the space provided. If you have to make an assumption then state what it was. Answers unaccompanied by supporting work will not receive full credit. The exam is closed book, closed notes, and “closed neighbors.” You are on your honor to have erased any course-relevant material from your calculator prior to the start of the test. Please print your initials at the top of each page in case the pages of your test get accidentally separated. You may separate the pages of the test if you like, and re-staple them when handing the test in.

Good luck!
Example System.

This example system is used for all of the questions in this test. A single system is being used to help reduce the time you spend absorbing system information rather than answering questions. However, this is a two-edged sword. Make sure that you really understand this example so you don’t make a systematic error across the entire test. Not every piece of information below is required for every problem. Even though the real-life system may not be implemented as a true distributed system, we’re going to assume that it is fully distributed for the purposes of this exam (i.e., every sensor/actuator has a processor, just like the elevator project you’ve been doing).

Consider the Hamerschlag Hall Pepsi Machine on the 1st floor. Assume that every button, light, and so on has its own processor (i.e., assume it is a fully distributed system in the fashion of the elevator used in the course project).

The following network performance constraints exist in this system. We suggest you skim the below bullets now and come back to understand them in detail when you need them to answer a question.

- The system uses a simple master/slave polling system for transmitting messages on an internal network, with a single network wire attached to all nodes in the system. Messages last exactly 10 msec each, and can be transmitted with no overhead (i.e., the system is capable of transmitting 100 messages/second). A master polling message and a slave message each counts as a separate 10 msec-long message (i.e., 20 msec for a master/slave message pair). A slave node transmits a slave message each time it is polled regardless of whether its transmit queue has real messages to send or not (i.e., if it has nothing to say, a slave node transmits a “reply – no real message to send” message in response to a master poll, consuming the entire 10 msec reply interval). There is neither drift nor jitter in any time-related aspect of this test problem.

- Messages are event-triggered, but sent according to master/slave polling. Each event triggers one and only one message. That means that a message is put into a transmission queue at each node for each event, and then sent at the rate of one queue message sent per poll received from the master.

- The ordering of messages within a round of polling varies, and is not known to designers – so you must assume the worst case in all message timing situations (additionally, there is no way to distinctly know when a particular “round” starts by observation, although the master node does keep track). The only assurance that is provided is that all slave nodes get a turn on the network within one round before any particular slave node gets a second turn.

NOTE: there is a potential race condition between the coin return switch and a vend operation. While this can be solved with a time delay, don’t worry about it for the purposes of this test (i.e., pretend no such race condition exists and do not account for it in your design).
High level requirements:

In general, this system must behave as the real machine behaves when working properly. When in doubt as to high-level requirements, make it act like a real vending machine. We’re going to make one simplifying assumption for the test: **assume that each can of soda costs $0.25, and the machine has hardware that automatically rejects any non-quarters inserted.**  (So, to simplify for this test, you only need to worry about quarters being inserted and all other coins are completely invisible as far as you are concerned.)

Specific items which you should be sure to address include:
- The machine shall vend a single can of soda for each quarter permanently retained.
- The machine shall return all deposited money if coin return is activated.
- The machine shall return all deposited money in excess of one quarter in any circumstance.
- The machine shall illuminate the light for a selected item while vending is in progress to indicate acceptance of a selection to the buyer.
- The machine shall illuminate the light for any out-of-stock item

System Sensors:

- **Button[s](v):** Soda selection button.  \(v=\{\text{True}, \text{False}\}\).
  One button per type of soda.  All are False at initialization.  \(S\) is an integer 1..8
  Button[s](True) is sent when button \(s\) is depressed; Button[s](False) is sent when button \(s\) is released.
  ***** Each button controller suppresses its own operation if the particular type of soda \(s\) is empty
  (i.e., Button[s](True) is never sent if Empty[s] is True.) *****
  * The button controllers as a group ensure that only one button indicates as True at a time; actually implementing this is not trivial… but is not on this test.  They do this by watching network messages – there is no special way for them to communicate to accomplish this.
- **Empty[s](v):** Item empty sensor.  \(v=\{\text{True}, \text{False}\}\).
  One empty sensor per type of soda vended.  True when out of stock.  \(S\) is an integer 1..8
  One per type of soda.  Initialized to be False.
- **Coin_in(v):** True when a quarter has been input to machine.  \(v=\{\text{True}, \text{False}\}\).
  Initialized to be False.  Note that the machine will permanently retain any inserted coin unless
  Coin_out(v) is activated (this is a simplification of real soda machine operation).
- **Coin_return(v):** True when coin return switch is activated.  \(v=\{\text{True}, \text{False}\}\).
  Initialized to be False.

System Actuators:

All actuators are assumed to “remember” their last commanded value and stay there unless commanded otherwise or forced otherwise by system/environment constraints.
- **Vend[s](v):** Vend soda of type \(s\) if True.  \(v=\{\text{True}, \text{False}\}\)
  One per type of soda.  When set to True vend a single can of soda.  \(S\) is an integer 1..8
- **Light[s](v):** Soda selection light.  \(v=\{\text{True}, \text{False}\}\).
  One per type of soda.  When set to True turns on the light in the button for soda \(s\); when set to False turns that light off.  \(S\) is an integer 1..8
  All lights set to False at initialization.
- **Coin_out(v):** When set to True, returns exactly one quarter.  \(v=\{\text{True}, \text{False}\}\).
  Initialized to be False.
1. **Coin Out Controller (25 points)**

Write the State, and Behaviors section for a coin return controller node whose purpose is to make sure that exactly the right number of coins are returned. (That is its only purpose – it is not responsible for deciding when to vend product.) You are not permitted to modify Input, Output, or Constraint information; you must provide for all constraints in your behaviors (i.e., you can’t make assumptions that some other part of the system will ensure a constraint is satisfied unless noted otherwise; but you can assume that all sensors/actuators are working properly).

**Input Interface:**
- Vend[s](v) (assume that any Vend message received indicates an actual vend event)
- Coin_in(v)
- Coin_return(v)
  (Empty[s](v) is purposefully NOT an input)

**Output Interface:**
- Coin_out(v)

**STATE:**

**CONSTRAINTS:**
1.1 Number of Coin_in events shall be greater than or equal to number of Coin_out events
1.2 Each Vend[s] event shall result in exactly one coin being permanently retained by the system. (It is permissible to assume that Vend[s] never occurs before money is deposited – preventing fraudulent vend requests is the responsibility of another object.)

**BEHAVIORS:**

(continue on back if necessary)
2. Vend Controller (25 points)

Write the State and Behaviors section for a vend controller node whose purpose is to dispense a correct can of soda consistent with meeting high-level system requirements. The same rules of answering the question apply as for Question 1. The system shall only vend appropriate cans of soda. ***The system shall vend a correct soda when the button is pressed and held while a quarter is inserted.***

Input Interface:
- Coin_in(v)
- Coin_return(v)
- Button[s](v)

Output Interface:
- Vend[s](v)

STATE:

CONSTRANTS:
2.1 A Vend[s] shall always be for a valid button press Button[s].
2.2 A Vend[s] shall only occur when a coin is available for consumption by the system.

BEHAVIORS:

(continue on back if necessary)
3. **State diagram (30 points)**
Pick either Coin Out Controller or Vend Controller and create a design at the StateChart level. (Precise use of statechart notation is not being graded – but you have to have an appropriate set of states and an appropriate set of transitions. Don’t forget initialization.)
4. Modeling & Global Time

4a) (10 points)
How long does it take for one round of messages to be transmitted on the bus for the complete system (i.e., for every possible transmitting message to be sent)? Assume that each sensor generated exactly one message per round, and that actuator output “messages” are actually physical actions that generate no message traffic (e.g., “Coin_out” in question 1 is a physical interface and not a network message – actuators do not generate message traffic, and thus are never polled). Also, assume there is a separate master node having the sole purpose of running the master/slave polling scheme that sends no messages except polling messages.

4b) (10 points)
Assume that there is zero latency involved in capturing a button press or button release and enqueuing the corresponding Button[s](v) message for network transmission. How far apart can two button presses be and still (worst case) have a situation where the second button pressed is the first button with the opportunity to transmit on the network? (Put another way, if you press two buttons, how far apart can they be and still have a chance that the second button will be the soda type that is actually vended). Remember that message order cannot be considered constant.