

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2005-05/msg00247.html>

- *From:* "H. S. Lahman" <h.lahman@xxxxxxxxxxx>
 - *Date:* Wed, 18 May 2005 19:06:40 GMT
-

Responding to Hansen...

Who's Greg? If we are doing tag teams I will have to get a partner.

You'll need more than one, because I have Me, Myself, and I working on this.

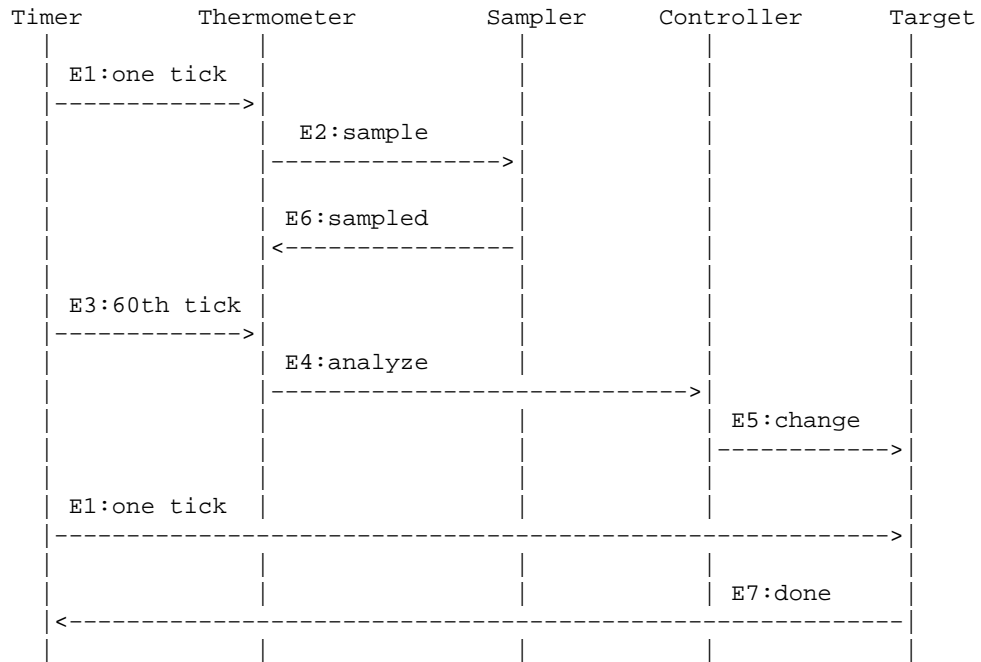
Ah, the old The Donald trick. I should have put that together. The mind is always the second thing to go.

I am arguing for eliminating SubstituteController because all it is really doing is delaying the Controller processing until Thermometer has the data ready. So let Thermometer announce to Controller that the data is ready directly and then let Controller do it thing with the data. Then there is no need for a delay because Thermometer issues the event when it is done -- whenever that is. [Any delay that emulates witching tick delays in the real controller is handled by using two threads whose priorities are tinkered with until the simulation provides a faithful emulation of skipped processing. However, as I discovered further below, that won't work. But as an academic exercise...]

A UML Sequence Diagram or Collaboration Diagram does a good job of representing events. Check it out in your new book. B-) What I am arguing for is essentially (assuming some content from previous missives):

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?



This is the Collaboration Diagram form where the sequence is not absolute in time (e.g., there are many E1 -> E2 -> E6 sequences for each E3 -> E4 -> E5 sequence). So Thermometer could get another E1 tick before Sampler generates the E6 event. Here Thermometer and Sampler run in different threads, as I assumed previously. So on a witching tick the slower Sampler thread will result in a skipped E1 tick sample.

Let's see if I can hang with this game. What I'd assumed before was that parts should be combined to reduce unnecessary units. So, for instance, I'd combined the germanium resistor, bridge circuit, AC supply, lock-in, analog to digital converter, and GPIB into a single Thermometer. Input is temperature, output is a voltage difference ($V=0$ at the setpoint).

Fascinating. I assumed it was the other way around; all that hardware you describe produced a voltage in a register that the Thermometer driver software converted to a temperature for the feedback (controller) processing.

Is there any software at all involved in producing the delta V? If not, then this doesn't seem interesting to the simulation because it will all be replaced by your heat flow calculation that produces the delta V. (I

Re: Lahman, how ya doing?

had that heat flow calculation living in Target and responding to the E1 event.)

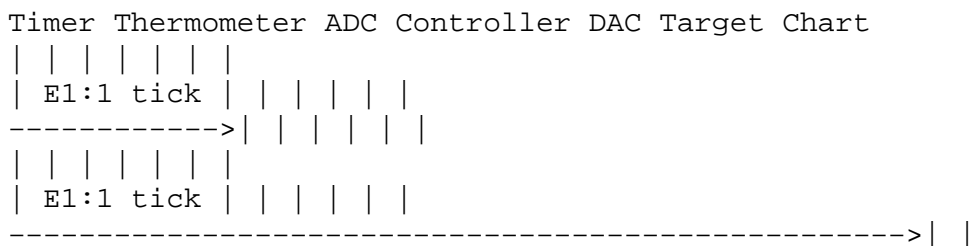
And I had the control logic, digital to analog converter, an attenuator, and heater resistor as Controller, with input being a voltage difference and output a heat. Technically everything after the DAC is analog, but it faithfully changes only when the controller tells it to change, there was no independent time evolution, and combining them let me get rid of a square root calculation, so it was as good as part of the controller. And the target itself was just a chunk of metal.

This sounds like a mix of software (control logic) and hardware (everything else). I was putting the control logic in Controller. I assumed the rest of the hardware is what the control logic controls. The hardware elements, like DACs and ADCs, would not appear in the simulation at all unless you had to explicitly substitute simulation computations (e.g., computing delta-V) to emulate them.

However, I get the sense below that Controller is limited to just talking to the stimulus hardware. That is, it is really Chart that is determining what the stimulus should do and Controller just knows how to relay that to hardware registers.

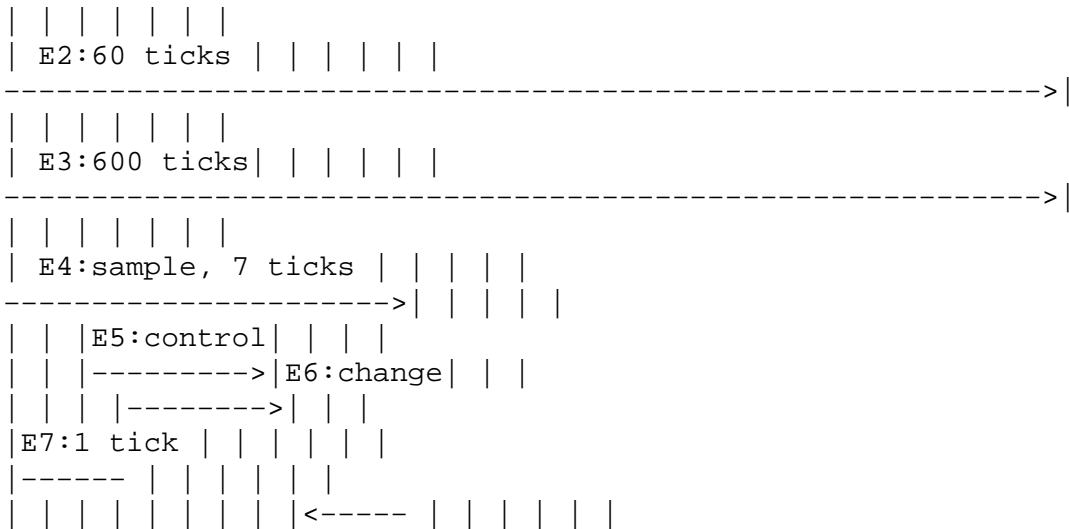
We could factor the ADC out of Thermometer (I believe that would be your Sampler), and factor the DAC out of Controller. Also, the controller gets a temperature every 7 ticks, a chart gets a temperature every 60 ticks. And the controller calculates a control action on a schedule, not on the command of the thermometer or ADC. In the non-OOP real program there's a big loop that includes a section that checks to see if it's time to do a control action. If so, it read a temperature and then calls the controller function. That's a lot like calling the Sampler on a schedule, and then the Sampler kicking Controller.

So let's see...



Re: Lahman, how ya doing?

Re: Lahman, how ya doing?



No events are going to Target or Thermometer other than E1, "analog time evolution", or being generated by them, because they have the role of passive elements that always do what they do, and always have their outputs ready for whoever wants to read them.

Confusion reigns. B-)

Why would a DAC or ADC be getting events? They aren't software in your simulation; they are pure hardware elements. At best some software element writes a value to a DAC register and you can't even do that for an ADC because the software can only talk to digital registers.

Also, I thought the 7/60th ticks were used to filter out 60 Hz pick-up. I assumed that was done in software but this suggests the hardware is making that adjustment. If so, that could be completely ignored because your simulation won't have any 60-cycle pickup in the delta-Vs because they are computed in the simulation.

Chart appeared a couple of messages ago, but I have no idea what it does. (You may have explained it the first time around but I've long forgotten most of that.) I assume it is whatever is doing FFTs and whatnot on the temperature samples. (More precisely, one a set of multiple statistical samples from each group of 60 temperature samples.) I also assume that once it accumulates 10 such samples Chart does something with them.

But I would expect that 'something' that Chart does would be to determine what the hardware does. So why isn't Chart talking to Controller? Similarly, why is Controller talking to a DAC rather than the Target? If that's just because the software digital view needs to be converted to analog for the hardware, that stuff isn't irrelevant to your simulation. You won't be writing to registers all; you'll be writing attributes that your hardware emulation software reads.

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

Finally, if the Thermometer always produces its temperature value on schedule, where are the skips? This segues to another issue...

The events you are showing seem to be mostly internal hardware synchronization (e.g., a DAC is triggered to read the value currently in its input register). That synchronization relies implicitly on the hardware design.

However, your simulation is going to trash that lock-step view because your simulation calculations are in the middle of things and they are unaccounted for by the hardware triggering scheme. Unlike the hardware, you can't count on things being done on a fixed schedule. Just because some processing starts at the right time doesn't mean it will finish at the right time. Where the hardware was designed to provide values in synch with the triggering, you are going to have to provide your own mechanism to do that since you are emulating the hardware. That will very likely mean some of your processing will have to wait until it is told things are ready rather than relying on a fixed tick scheme.

Since I clearly don't understand what the real controller is doing after all this time, I am not sure it is worth going into much more explanation. However, I would strongly advise separating out what is hardware and what is software in the real system. Once that is done it will be a lot easier to plug in simulation software for the hardware elements while reusing the software elements. I think it will also make it easier to see where you can synchronize on a fixed schedule and where you need to daisy-chain events.

I am also assuming Controller is doing the heavyweight data analysis and Target just emulates the initial response to whatever Controller has defined as the new stimuli state. I also assume Target is the front end for the heat flow calculations needed to produce a temperature value on each tick in the simulation. So Target also needs to get an E1 event from Timer to trigger that computation. When that <presumably lengthy> computation is done, one is ready for another Timer tick so Target sends the E7 event (as an alternative to a self-directed event by Timer).

The calculation is actually good for a single line of arithmetic. Heat pulses travel surprisingly quickly in this low temperature regime due to low heat capacity and high thermal conductivity, so the operating equation is just

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

$\Delta Q = C \Delta T$

OK, it isn't the sort of diffusion computation I had in mind. But you still have multiple "blocks" to propagate through and for each block you have to convert back from Q to T to provide input to the next block. You may have some housekeeping to do for old vs. new as well. All in all, I bet it will take a lot more real time that the hardware takes to provide a value.

In addition, your description above suggests you are going to have to emulate more of the hardware than I thought (e.g., if the 60-cycle filter is done in hardware). Rather than sorting that all out in a lock-step trigger schedule as the hardware does, it might be a lot easier to work around it by daisy chaining events.

[I know I argued you should be as faithful as possible to the hardware and you should. But at some point the simulation considerations start to screw things up and you have to raise the level of abstraction to what the hardware triggering was trying to accomplish and emulate that rather than the specific mechanism.]

OTOH, running all the hardware emulation stuff in a high priority thread may work. B-)

(1) Insert a "delay" between whoever generates the normally scheduled event and whoever responds with the processing that must be skipped for the current tick. In effect the delay is such that when the incoming event is re-dispatched it is in the next tick. (This was your first feedback loop diagram.) I think finding a reasonable mechanism for the delay would be tricky, given the disparities in processing time. It also gets a bit hoaky because the "delay" may get another tickle in the next tick and it should not double dispatch in that tick. In that situation one is essentially doing (4) below.

What it should do would depend on the details. But if there was a Delay catching sample events, it only needs one vacancy and then can discard any following sample events until the first is dispatched. Other events, like reset, are sort of meta-simulation and should be passed immediately.

By delay mechanism, I mean postponing processing of the event until the next tick. To do that you have to know when the next tick is in order to

Re: Lahman, how ya doing?

Re: Lahman, how ya doing?

re-dispatch it at the right time (i.e., put the event addressed to the right target on the queue at the right time).

There is nothing wrong with me that could not be cured by a capful of Drano.

H. S. Lahman
hsl@xxxxxxxxxxxxxxxxxxxxx
Pathfinder Solutions -- Put MDA to Work
<http://www.pathfindermda.com>
blog: <http://pathfinderpeople.blogspot.com/hslahman>
(888)OOA-PATH

Re: Lahman, how ya doing?