

T. H. E. SOLUTION

Product Development Consulting

A DESIGN APPROACH FOR COMMERCIAL REAL-TIME EMBEDDED SYSTEMS

INTRODUCTION

Most commercial embedded products are not concerned with life/safety critical events and therefore do not fall into the "hard" real-time arena. They do typically have to incorporate a time dimension in their specification, design and implementation since they include time critical functions such as desired response times. We might term these "firm" real-time systems since there is potentially a significant price to pay for less than optimal performance in terms of sales.

Since most commercial embedded products are sold in thousands or tens of thousands any fixed cost, such as software development, is quickly amortized. What impacts each and every product sold is the parts that go into manufacturing it. The result, the predominate economic impact is in the hardware selected. This can result in severe restriction on processing power, memory etc. This has to be balanced with the requirement for optimal performance.

The remainder of this article will examine the issue of capturing the time dimension when designing commercial firm real-time embedded products. The method discussed uses only tools that should be available to any engineer. The design of an intelligent tape drive controller will be used to illustrate the method.

PERFORMANCE MODELING

The mechanism that allows us to capture time can be termed performance modeling. The generic process of building a system performance model can be described in six steps. These are:

1. Identify the system performance parameter(s) of concern.
2. Build a basic analytical model to focus on key issues.
3. Gather information required implementing the model. Sources include:
 - a. Published material
 - b. Prior systems
 - c. Prototype implementations
4. Implement the model using either analytical or discrete event simulations.
5. Run the model and analyze the results.
6. Continue steps 4 and 5 until you are satisfied.

The following section will apply this generic process throughout the software development cycle.

AN EXAMPLE

The example we will use is an intelligent buffered tape drive controller. The purpose of the controller is to provide the link needed between the "raw" streaming tape device and a host computer. Buffered controllers are intended to optimize the performance of the tape system by buffering data transfers. This allows the host to operate asynchronously of the tape device and at substantially different burst rates. When one considers the substantial throughput loss that occurs when a streaming tape drive has to reposition, the role of the controller becomes one of significant consequence.

Tape drives have two primary parameters that can be used to characterize system performance. The first is host perceived throughput. The second is the record size required to maintain streaming conditions. Since streaming tape drives are primarily used for backup or archiving type operations these parameters are normally considered for the "more important" operation of writing data to tape.

If the controller's overhead is not a factor, host perceived throughput is controlled primarily by the proper sizing of the data buffer on the controller. If the buffer is equal or greater than the maximum amount of data that could be transferred during a reposition, then the host will perceive either the bandwidth of the tape or its own, whichever is least. Since this is the best that can be done on average there is no need to explore this parameter further.

The second parameter, record size to maintain streaming, captures the point where the controller's overhead affects host perceived throughput. Typically we want this as small as possible at least as small as the normally expected record/block size used by the host's operating system. In addition this parameter is important due to the adverse effect of reposition on the tape device. These effects include mechanical wear etc. In general under any given host the tape system that will fair the best is the one that has to do the least number of repositions. This parameter has no simple answer and deserves further exploration.

The requirements definition phase is concerned with the transformation of vague requests and specification from clients, marketing etc. into a form which clearly and accurately defines what the software system must provide. The requirement definition must include a definitive specification of the performance of the system. This is typically handled with a simple analytical model. In our example the dynamics of the situation can be easily described with the input/output model diagrammed in Figure 1.

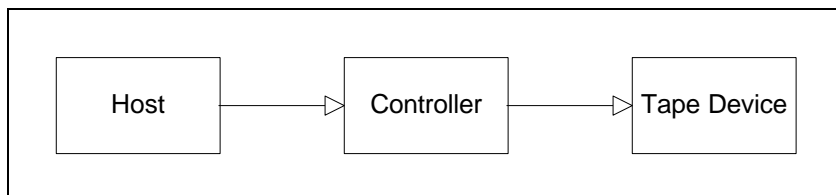


Figure 1, Input/Output Model

The controller is nothing more than a FIFO pipe with its input characteristics set by the host interface and its output characteristics set by the tape device interface. When we are just streaming the input time per record and the output time per record must be the same. The output

time per record is:

record size (RS)/ device data burst rate (DDBR) + time for inter-block gap (TIBG)

The input time per record can be similarly defined as:

record size (RS)/ host data burst rate (HDBR) + time between commands + processing time (PT)

The time between commands is nothing more than the host processing time not "covered" by the data transfer time. If we assume a dedicated "optimal" host, i.e. a host that is ready to send the next record as soon as the tape system is ready for it, this time is zero. As a result we get:

$$RS/DDBR + TIBG = RS/HDBR + PT$$

Since everything but record size and processing time are known characteristics of the host or device we get:

$$RS = C1 * PT - C2$$

If we know the controller's processing overhead, as perceived by the host, we can predict the streaming record size.

The basic analytical model has indicated that we need to know three parameters that are characteristic of the host and tape device, DDBR, TIBG and HDBR. These are readily available dependent on the host interface supported, tape format etc. The requirement for the controller's processing time is not as easily obtained. We need a model of the software. Figure 2 provides a preliminary high level data flow description of the software system that is expected to be adapted. This provides the principal flow of the system and identifies key synchronization / communication points. Each of the transforms identified will be a task in a multi-tasking system. The data stores shared between the tasks will be implemented as monitors. Since the occurrence of errors while trying to transfer data to tape is rare and normally forces a reposition, we will assume an error free environment.

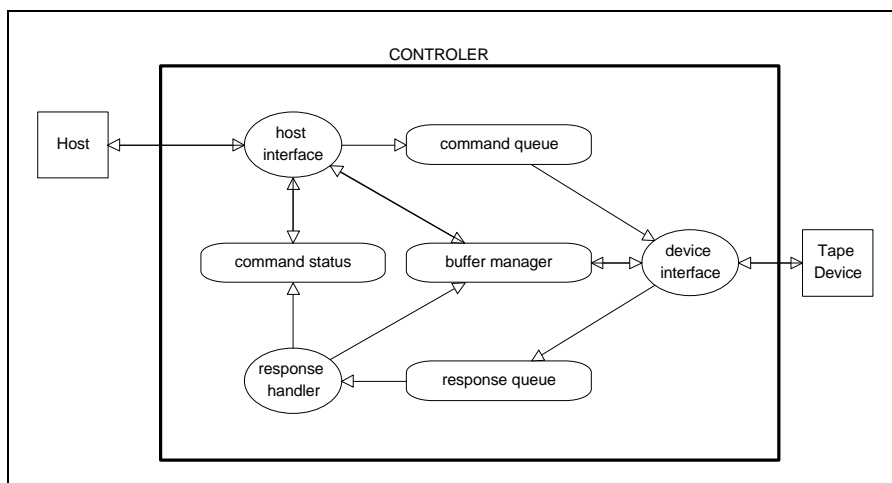


Figure 2, Data Flow Diagram

Per the initial discussion in this section, we are only concerned with the write operation. The basic operation of each task for the activity under consideration can be outlined using very high level pseudo-code. A partial example of this follows:

host interface task ()

```
{
  initialization;
  while (FOREVER)
  {
    wait_event(command available);
    if (no error on prior command posted)
    {
      switch (type command)
      {
        case WRITE: allocate(record size);
          set up data transfer;
          wait_event(transfer complete);
          send response to host;
          build generic write command;
          queue_cmd(write command);
          break;
        :
        :
      }
    }
  }
}
```

At this point the next move depends on our current knowledge. For instance if we had implemented a similar system with a similar real-time executive etc. we could build execution time estimates for the pseudo-code from this knowledge. If we had no experience with this type of software system, processor type etc., we could build a small software system that captured the key flow and synchronization features but "stubbed out" the interfaces and used timed loops to replace most of the code. We would run this performance prototype on the target if available or a general purpose computing platform that used a similar processor. We would end up with a per task "timing" chart. At its least resolution this chart would look something like the following:

Host interface task:

- processing prior to data transfer (H1)
- data transfer
- processing after data transfer (H2)

Device interface task:

- processing prior to issuing command to device (D1)
- data transfer
- read after write delay (D2)
- processing after receiving response from device (D3)

Response handling task:

processing to dequeue response and "throw away" (R)

Whether we are using estimates based on prior experience or performance prototypes, the purpose of gathering the information is to implement the model in a simulation. The simulation can be either an analytical one or a discrete event. The choice of one or the other depends on how dynamic the simulation needs to be. Discrete event simulations allow one to build very detailed and dynamic simulations. They can capture task switching, queue sizing etc. In our example we just want a feel for the streaming record size. We can use an analytical solution if we are willing to accept assumptions that are fairly static. Assumptions would include the requirement for a single fixed record size and no file marks. Since we are looking at things when we are just short of repositioning, all the processing in the critical path must be completed on a per record basis. It makes no difference in what order. Further, both best and worst case processing time (from the host's perception) can be quantified with the estimates we made in the last sub-section. The worst possible case is that none of the device interface processing occurs during the data transfer on the host side. The best case processing is that all the device interface processing that can occur during the host side data transfer does. In terms of our time chart this becomes:

Worst case: $PT = H1 + H2 + D1 + D2 + R$

Best case: $PT = H1 + H2 + D1 + D2 - RS/HDBR + R$
where $D1 + D2 - RS/HDBR \geq 0$

Given values for the execution times we can solve for record size. This record size, along with the assumptions that allowed it to be estimated, would provide our performance specification.

The design phase is concerned with the translation of the requirements into a specification of sufficient detail to allow for correct code to be generated. During this phase we would normally build on the basic model described for the requirements phase. At this point we are interested in progressively more detail. We would like to create a system model that will allow us to capture the true dynamics of the system. This would allow us to set a time budget and to determine the sizing of shared data structures. Typically this can be achieved using a discrete event simulation. In these simulations the synergistic result of hardware and software is execution times, time delays waiting for some external action to take place and contention over resources, such as the CPU itself. Though a large number of simulation language are available now, good results have been obtained using one of the original languages, GPSS. This language directly incorporates such concepts as priority driven task preemption and as a result can be easily used to build realistic models.

The level of modeling depends on the complexity of the system and the level of realism that must be captured, error handling included etc. The minimum acceptable data that must be provided by the model is sub-task level time budgets and sizing of all data stores shared by the tasks. A example of a time budget, in the form of a task timing diagram follows:

interjected into the model to determine system impacts and revised performance specifications.

CONCLUSION

The proper specification, design and implementation of commercial embedded products must normally include the dimension of time. This is true even when the function being performed is not life/safety critical. This "firm" real-time problem can be correctly handled with the use of performance models. The models may be analytical or discrete event simulations depending on the stage of development and the amount of dynamics and detail required. In any case the basic implementation tool is the time budget. With the aid of performance modeling techniques this budget is established and refined. During implementation actual performance vs. budget is used to monitor for "correct" implementation.