

Modeling Human Factors with VHDL

Todd P. Carpenter and Dr. Chris A. Miller
Honeywell Technology Center
3660 Technology Drive
Minneapolis MN 554181006
carpent@src.honeywell.com
cmiller@src.honeywell.com

Abstract

This paper describes an approach to using VHDL for human factors modeling, specifically the interaction of an operator with a complex computer system. Ordinarily, such analyses are performed using simulation languages and tools explicitly formulated for this purpose. The tight integration with detailed system design practices and the availability of numerous VHDL support tools suggest significant benefits for using VHDL in such analyses.

1 Introduction

Successful development and application of computationally demanding, complex, high performance electronic systems relies heavily on complete understanding of the system requirements. The integration of human interaction and capabilities is an integral facet of the design and development of critical systems in space and avionics applications. The increasing complexity of large scale systems makes prototyping prohibitively expensive. Inaccuracies in analytically predicted behavior of these systems are often too costly to accommodate after system implementation. Therefore, we are examining the potential for combining models of both the system under development as well as of the human operator to gain relevant feedback earlier than is otherwise possible using conventional techniques.

This paper documents an application of VHDL to modeling the interaction of a human operator with a complex computer system. Specifically, we will model the task loading incurred by a human operator in attempting to use the system. Such analyses are typically performed using simulation languages and tools explicitly formulated for this purpose, notably the Systems Analysis of Integrated Networks of Tasks (SAINT). Our motivation here is to demonstrate that VHDL, while a hardware systems modeling language, can be easily adapted to provide some of the capabilities required for human factors modeling at the task level. This provides evidence that some single modeling framework might eventually be capable of integrating the many disciplines necessary for complete systems design and development.

The paper briefly describes a layman's perspective on a human operability model and the metrics associated with the analysis of it. Next, the techniques and tools used for implementing such a model in VHDL will be detailed. An example system is then developed and analyzed. Finally, some pitfalls of such interaction, most notably the dynamic range of the system including the human operator, are discussed. The focus of this paper is on the VHDL implementation of a model that would generally be considered far outside the normal domain of VHDL.

2 Human Resource Model

When developing high-performance electronics systems that must interact with humans, numerous questions must be answered about the man-machine interfaces. For instance, at what rates will the operator be required to interact with the system, how busy that operator will be, and how long will it take that operator to respond to the various attention demands. Examples of applications where the answers to such questions are important range from high performance and esoteric military consoles, to cash registers at the everyday supermarket. In either case, operator overload can result in missed deadlines, which then have varying degrees of consequences from costly to catastrophic.

These operator performance issues are analogous to those which arise in the development of the actual electronics system. A variety of special-purpose tools and techniques have been created in the human factors field to model human performance, workload, and resource usage. This paper describes how the same techniques we use in hardware design can be applied to modeling the man-machine interface. We will illustrate how human factors modeling used in one such tool [2] can be implemented in VHDL. This

Linear Process

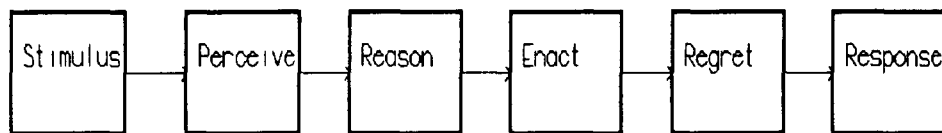


Figure 1: Example of simple process flow model

integration of human performance and system performance in a common language has the potential for allowing easier interaction between the hardware realm and the human factors, as well as establishing and validating real system requirements.

A *model* is a representation of some system. Typical models include textual specification and requirements documents, analytical models, simulation models, and physical models. A model is typically used to describe the system in something other than the actual physical realization. Analytical, simulation and physical models can be used to predict behavior of the system. One might wish to use models instead of the real system for a number of reasons. One frequent use is during the design phase of the system, when the actual device is not yet available. Other uses include those cases where live testing would be too slow, costly, or destructive.

Throughout the course of this paper, we will develop an model for the interaction of a human operator with a complex electronics workstation. For this particular situation, the model itself, and not the particular values, are the important parts of the experiment. Furthermore, to simplify matters, we will concentrate fully on the model of the human interaction, and ignore the inner workings of the workstation, assuming that the reader is well versed in VHDL modeling and simulation of electronics systems. At the end, we will summarize and provide some insight on problems which might arise when including detailed electronic models.

Introduction of Example

An example of a very generic task network is shown in Figure 1, and is representative of human tasks which might be performed in conjunction with a workstation. This model is presented here to provide a frame of reference, and will be used later in the paper for an analysis of timing, and workload issues. The flow for this process is from the left to right, and is described as follows. Initially, stimulus is provided to the operator, at a rate and type indicated by the *stimulus* block on the left. This stimulus is then perceived by the subsequent *perceive* block. After perception comes some reasoning about that which was perceived, followed by some physical action modeled by the *enact* node. After the physical action we have a node labelled regret, where we are modeling potential operator regrets over the previous action. Finally, we receive the final output of this process in the node marked *response*.

Metrics

One of the first things one must do when modeling any object is to determine the objective of the modeling activity. These goals will direct the fidelity and composition of the models that are developed. We desire an understanding of the different, time varying demands placed upon an operator, which we will call the time dependent workload utilization. We also desire a quantitative measure of the time it takes for the operator to perform a given task, which is the task latency. These metrics are to be generated from a simulation of the operator performing a specific task. To accomplish such measurement, we will obviously need to capture the flow of activities. And for each activity, we must be able to quantify the demands placed upon the operator, as well as the length of time it takes for that activity.

Resource Channels

The human operator can generally be described in terms of resource channels. A particular activity will require a certain level of application of particular channels. Humans can concurrently handle multiple demands on the same or different channels. However, those concurrent demands do interact in varying degrees, resulting in different operator performance. At a high level, we can conceive of these channels as

being broadly categorized into three major groupings of perception, cognition, and psycho-motive. These three channels can then be broken down into further sub-channels. The following discussion will describe typical resource channels as well as a description of the interaction of demands upon those channels.

Perception

Perception is the activity that results in the awareness of the elements of environment through physical sensation. Such observation generally includes the common five senses, auditory, tactile, visual, olfactory, and gustatory (taste). For the purposes of this paper, we will concentrate on perception through the former three senses, since we do not currently conceive of high-performance man-machine interfaces operating through the latter senses.

Cognition

Cognition is the act or process of thinking or knowing, including both awareness and judgment. Cognition has many sub-categories, involving accessing memories, understanding perceptions, judging or forming opinions, etc. We will consider three primary aspects of cognition, namely understanding the spoken word, and interpreting its meaning (i.e., verbal reasoning), forming opinions, reasoning, and judgement (i.e., analytic reasoning), thinking pertaining to the position or location of objects (i.e., spatial-reasoning).

Psycho-motor

Psycho-motor resources relate to muscular action believed to ensue from prior conscious mental activity. These would be the directed actions that the operator performs as a result of earlier perception and subsequent cognition. This activity can have numerous physical manifestations, such as vocalization, locomotion, and manual interaction. Vocalization is the audible generation of commands or thoughts. Locomotion is directed muscular activity with the goal of changing the current position of the operator. This includes reorientation, such as changing the perspective by which something is viewed or changing the focus and direction of the eyes. Psycho-motor also includes relocation, such as moving the position of the operator from one point to another point, via walking, pushing about by the arms, etc. Manual interaction is the effort by the operator to change the external environment, such as activating switches or moving objects.

Interaction of Resources

Workload for some task or set of tasks may be modelled by reducing the operator demands on each of the above resource channels to a single scalar value. Workload consists of three components. The first component of workload is the sum of the demands on each of the various resource channels.

The second workload component is the result of loading on the different channels, and the relative weights of those channels. For instance, we earlier identified perception, cognition, and psycho-motor channels. We can easily understand that perception demands do not require the same level of interaction as other resource demands, such as cognition. For example, we are used to seeing much and assimilating many different things at the same time. However, when thinking, we rarely reason about different things at the same time. So multiple demands on a cognition channel will have a higher performance penalty than multiple demands on a perception channel.

Demands placed on the various resource channels will also interact with each other, which produces our third workload component. Consider a person doing a specific job, for instance, reading a book. That person would have a particular reading and comprehension rate, based on the complexity of the subject matter being read and assimilated. That same person might also be able to participate in a conversation, and answer complex questions at some rate dependent upon the nature of the question. If however, the person were required to read the book *and* answer questions of an unrelated topic at the same time, the reading rate and comprehension rate can decrease. Moreover, it is reasonable to assume that the performance would decay with the multiple activities since time would be lost doing the equivalent of a context switch. For instance, if normally the reading rate is one page per minute, and the rate at which questions can be answered is two questions per minute, it is not reasonable to assume that if both activities are ongoing that the both rates are simply halved. Rather, the reading rate might now be one page every three minutes, and one question a minute.

Finally, we will sum these three workload components to form our overall workload measure. This workload value will be tracked, so we can see how the overall demands vary as a function of time. This workload value may also be used to indicate how long particular activities take, by acting as a multiplicative factor on the nominal length of time for that activity. These equations can be seen in the following:

$$\begin{aligned}
W &= W_1 + W_2 + W_3 & (1) \\
W_1 &= \sum_{\forall i \in Demands} i \\
W_2 &= \sum_{\forall i \in Channel} Demand_i \times Weight \\
W_3 &= \sum_{\forall i \in Channel} \sum_{\forall j \in Channel} Conflict_{ij} \times Demand_j
\end{aligned}$$

External experimentation must be used to establish the values for these conflict and demand values. We will not address the specifics of those numbers or the way in which they are derived in this paper.

3 VHDL Implementation

Given the above human resource model, we need to develop an environment that allows rapid, straightforward, and maintainable capture and evaluation of a human factors model. Such an environment must be flexible, so that numerous, diverse scenarios can be modeled. It must at the same time be a natural representation, so that the human factors expert need not also be an expert programmer in the chosen representation. In addition, the modeling environment must permit easy access to the state of the model, so that any desired runtime behavior can be extracted in a straightforward fashion. Finally, any such representation must be maintainable, so that other models may interface with it, as well as be supported and usable in future experiments.

Graphical Approach

Examining the above criteria, we see that ease of modeling is a primary concern. Particularly, we do not want the human factors engineer to have to delve too deeply into the inner workings of VHDL. However, we also want to create a model that is flexible enough to allow the engineer a wide range of expression capabilities. However, VHDL coupled with a powerful graphical interface permits such expression.

Graphics allow the designer, modeller, and evaluator all the same access to the flow and representation of the model. Required facets of a successful graphical representation include access to the underlying signal types, connectivity, attributes, behavior, and other modeling characteristics, preferably in some linked, straightforward fashion such as via a hyper-text environment. While various editors come close to fulfilling some of these requirements, few are able to provide the complete desired functionality. We await the advent of an EDIF graphical standard, coupled with such tools. Meanwhile, we have the choice to avoid graphical entry mechanisms since they are not perfect, or glean what benefit we can from what is currently available. We have chosen that latter approach, since so much is already available. Our pay-back is multi-fold. The team is more effectively able to communicate design ideas and status, since the design structure is graphically captured, and behavior is textually described. Furthermore, we design our modeling capability around the idea of graphical entry, so the human factors engineer need not be overly concerned about the underlying representation, but rather, will be able to think in terms of behavioral components and attributes on those components. This can significantly reduce the time and expense of bringing such modeling capability to the different disciplines.

With this in mind, we have developed basic components which will form the foundation for modeling higher-level processes. Our goal was to make these sufficiently powerful and configurable such that arbitrarily complex functions may be modelled with as few nodes as possible, while at the same time, not making the nodes so complex that performance and ease of use is lost. We wish to allow the designer to graphically capture the flow of the process by simply placing nodes on a display, and connecting them to show the flow of events, and immediately begin simulation and evaluation. After that point, further interaction would allow the designer to add more detail and characterization to the models.

Token Based Communication

The architecture of the man-machine system is captured as a structural model of interconnected processes or activities, where each activity represents some atomic action by the operator. Control or dataflow is accomplished by the interconnection via signals of an atomic type called token. Since these models are not intended to address actual system functionality, at least insofar as that real data is not processed and real

| Field | Description |
|-------------|--|
| id | Global ID of token to differentiate between them |
| priority | Explicitly token priority for resolution |
| size | Amount of work represented by token |
| start time | time token was initially created |
| destination | final destination of token, for control flow |
| source | source of token for tracking purposes |

Table 1: VHDL Token Fields

outputs are not generated. The data is characterized and represented symbolically via tokens. These tokens represent some quantum of work that must be accomplished, and will carry some indication of the type and amount of work that must be performed.

The token is a resolved signal which is a record type that has the fields described in table 1. The size field is the primary method by which data dependent behavior is controlled. Generally, a node's latency will be a function of the amount of input data. This field indicates how much information is to be processed. The destination field is the primary control flow operator. All the nodes examine their inputs, and if a token has a destination field that is a wild-card match with the name of the node, then that token is accepted and the node behavior commences. This functionality was chosen since it allows multi-drive and multi-drop interconnections. Many previous modeling tools required point-to-point connections, and we discovered that such requirements were extremely cumbersome, since we found ourselves frequently adding and deleting nodes as the design process progressed. Being able to add a node and simply hook it's input and outputs to existing nodes is a great time saver. However it adds the complexity that the destination must be specified and some technique established for the subsequent routing. However, this is all done internally by the library packages, and does not need to be visible to the user. The token resolution function is a simple priority scheme that weighs each of the fields in turn, and as soon as one token is found greater than the other, it wins the arbitration.

Generics

We choose the VHDL Generic as the means by which the designer would interact with the model to provide individual component characterizations. Generics are a VHDL capability precisely targeted to such customization. We have successfully used a wide variety of generics in the past on electronics systems performance models, and this is a natural extension of the same technique.

Each node in the system has a number of generics attached which control how that node interacts with its environment. The generics used for human factors modelings fall in three categories. They are statistic, control flow or functional, and resource generics. The first set of generics, the statistics, control how information is reported about each particular node. They include such things as the node name and whether or not internal tracing is enabled on that particular node. The control or data flow generics model the functional behavior of the node. They control when the node fires, the amount of data produced by the node, the nominal length of time the node will be busy processing the input data, and the destination of the output data. The last set of generics model the workload demands placed upon the human resource channels. The model generics can be seen in Table 2.

All of the generic numeric parameters are described via a Generic Description Language (GDL). GDL is a language used to deterministically or stochastically specify numbers or strings. The original syntax was developed by Honeywell as part of our ADAS modeling environment, and adapted for VHDL under the Graphics Processor Description (GPD) Program out of Wright Laboratory. Currently supported distributions can be seen in Table 3. These distributions can be constrained via a "RANGE" modifier to keep their values within reasonable bounds. Each distribution may be seeded to maintain uncorrelated random value streams.

For example, we might have a perception node with a nominal latency of 100 milliseconds, and the simple

| | |
|---------------------|--------------------------------|
| Statistical | |
| TRACE | controls information reporting |
| REFERENCE | documentation tracer |
| INST | hierarchical instance name |
| Control flow | |
| LATENCY | nominal component delay |
| DESTINATION | output destination for data |
| SINKANY | controls input data acceptance |
| UNIT | basic node atomicity |
| TXFORM | data expansion ratio |
| PRIORITY | output data priority |
| Resources | |
| Auditory | Perception |
| Tactile | Perception |
| Visual | Perception |
| Verbal | Cognition |
| Analytic | Cognition |
| Spatial | Cognition |
| Manual | Psychomotor |
| Vocal | Psychomotor |
| OtherResources | Psychomotor |

Table 2: VHDL Node Characterization Generics

| Distribution | Description |
|---------------------|--|
| COUNTER | repeatedly count from 0 to n |
| CONSTANT | constant value |
| UNIFORM | uniform random distribution |
| EXPONENTIAL | exponential distribution with specified mean |
| GAUSSIAN | Gaussian (normal) distribution with specified mean and standard deviation. |
| POISSON | Poisson distribution with specified mean |

Table 3: Supported GDL Distributions

```

GENERIC latency : STRING :=
    " GAUSSIAN 100 15" &
    " RANGE 50 150" &
    " SEED 17 67345";

```

Figure 2: Example of a Generic Specification

GDL description would be “CONSTANT 100.” However, lab experimentation would tend to indicate that the time for perception actually follows a normal distribution, so a more accurate specification would perhaps be “GAUSSIAN 100 15 RANGE 50 150,” where we have also constrained the minimum and maximum possible values that will be returned. Finally, we can also seed the random number generator with a number from a random number handbook and we can end up with a complete specification as shown in Figure 2.

Global Workload Resource

We now must determine precisely how workload will be implemented in VHDL. At first glance, we might be tempted to make a node specifically for workload determination. When each node began, it could send a request to the workload determination node which would then consider that demand with the rest of the concurrent demands. A value could then be returned to the requesting node, and that value would be used to determine its latency.

Unfortunately, our earlier modeling requirements that this modeling environment be straightforward and natural to use precludes this approach. The purpose of building this modeling environment is specifically for workload determination and the nodes in our model will represent activities placing demands upon the operator. Having a node represent the workload determination function seems rather unclear. We would need explicit signals to connect that node to the other nodes in the process activity. We would also need to somehow handle the different number of ports on the workload determination node, dependent upon how many process activities there were. All of this seems rather cumbersome, and at the very least will serve to clutter any graphical or structural representation.

However, we do have a powerful signal based technique that serves perfectly. It is a VHDL global signal and a bus resolution function. The global nature of the signal allows us to add or delete activity nodes in the process, without being concerned with how they hook into the workload resource model. The bus resolution also allows us to generate a model that does not have any significant modeling artifacts in it.

This global signal will be called Workload, an array indexed by the various resource channels, where the value of each element represents the demand placed upon that channel. The bus resolution function is very simple and fairly directly implements the workload equation 1, but suffers in that it is non-associative. The resulting workload value, along with the maximum workload thus far encountered is stored in another field in the workload signal.

Modeling Nodes

For this initial modeling environment we decided to provide three basic components. These are the Input, Output, and Process nodes. The Input component does not have resource channel demands associated with it, but is used purely to generate stimulus into the rest of the model. An Input Node example would be the *stimulus* block in Figure 1. The Output component also does not have resource demands associated with it, but simply consumes tokens generated from the rest of the model. An example of the Output node is the *response* block in Figure 1.

The main modeling workhorse is then obviously the process node. This component accepts data, calculates its workload demands, determines its internal delay based upon the global workload value, and generates an output token once that delay has expired. All the nodes between the stimulus and response blocks in Figure 1 are instances of this Process node. The individual characterization of each of these nodes is what forces them to behave according to their intended purposes.

Parallel Process

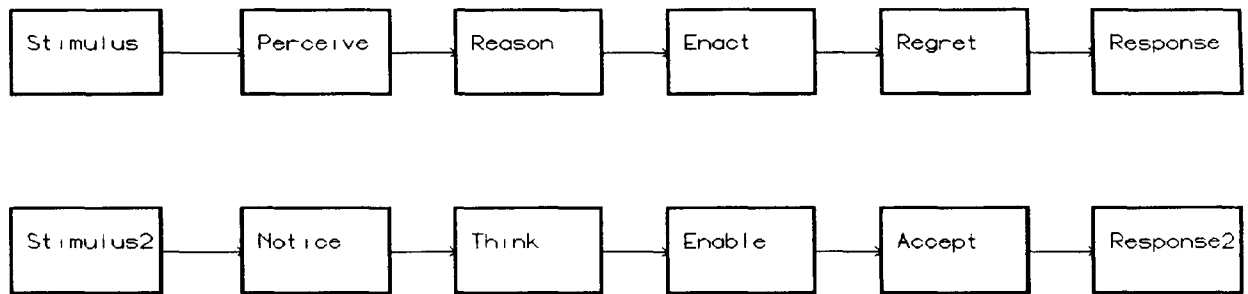


Figure 3: Example of parallel process flow model

4 Results of Experiment

The models discussed in this paper are purposefully generic and simple to make it clear how the models operate. The model and its behavior are more important than the particular values, so we will not be concentrating on an analysis of the values. In addition to the linear process we showed in Figure 1, we also built a model showing how parallel, unrelated activities interact with each other. This model can be seen in Figure 3, where the upper sequence of activities is identical to the previous sequence. The lower sequence is a similar flow of stimulus and response actions, however it is a higher rate process. The parallel model of Figure 3 by no means implies that the only activities that can be modeled include uncorrelated processes. We can easily connect the two chains, have cycles, decision points, etc.

The following sections cover the results obtained from modeling the simple, linear operator task, as well as the parallel, uncorrelated activity. The graphs shown here are postscript files generated off of VHDL `textio` output. We have examined different techniques for reporting pertinent simulation information, including real-time display of information, vendor supplied toolkits, and post processing of `textio` generated by the simulation models. All things considered, this latter options remains the most portable, flexible, and powerful solution we have discovered, especially since such “post processing” in a multi-tasking system can nearly provide the equivalent to real time processing and display. With this in mind, we are using standard Unix tools, with a strong desire to not have to recompile or port tools to the different Unix architectures.

All the models generate information about when they fire, for how long, and the values of the tokens that are passing through them. This information can be easily filtered into columns of data, which can then be converted to numerous display formats. In this case we have written some short *awk* scripts which generate the following postscript charts:

activity charts show when a particular processes is active.

workload charts show the global workload as a function of time

latency charts are a histogram of the critical path latencies

The first two figures, 4 and 5 show us when the different nodes in the two different models are active. The bars show when a particular process is active in time. The numbers on the right are the overall utilization (time busy/total time) for that activity. To reduce the complexity of the results and to make the interaction clear, we have modeled all attributes as constant values. The interesting experiments would come with stochastic distributions, but that plainly makes the interpretation more difficult. In the first activity chart we see that each node is active in succession. We carefully insured that only one node was actively firing at any time. Therefore, we have no second order effects on the delays of the nodes due to concurrent workload demands. This is obvious when we examine the histogram of the path latencies in Figure 6, where we see all the latencies are identical. If we introduce another process that makes concurrent resource demands, as in Figure 3, the length of time to complete a particular activity is no longer constant but depends on the current value of the global workload. If we look at the histogram of latencies for this parallel model in Figure 7, we see two major clusters of latencies. The cluster on the left represents the latencies through the faster process. The cluster on the right represents the time to complete the slower process. However, notice that the times are no longer bunched all at the same rate, but are spread out.

Finally, we can examine the workload as a function of time. These can be seen in Figures 8 and 9. Again, we notice the linear process with no concurrent workload demands behaves very regularly. When we add conflicting resource requests we notice that the workload immediately takes on a significantly noisier aspect. We also notice that the maximum height of the workload (43) rises significantly from the previous maximum (14). While we again stress that the number themselves are not significant in this case, the relative

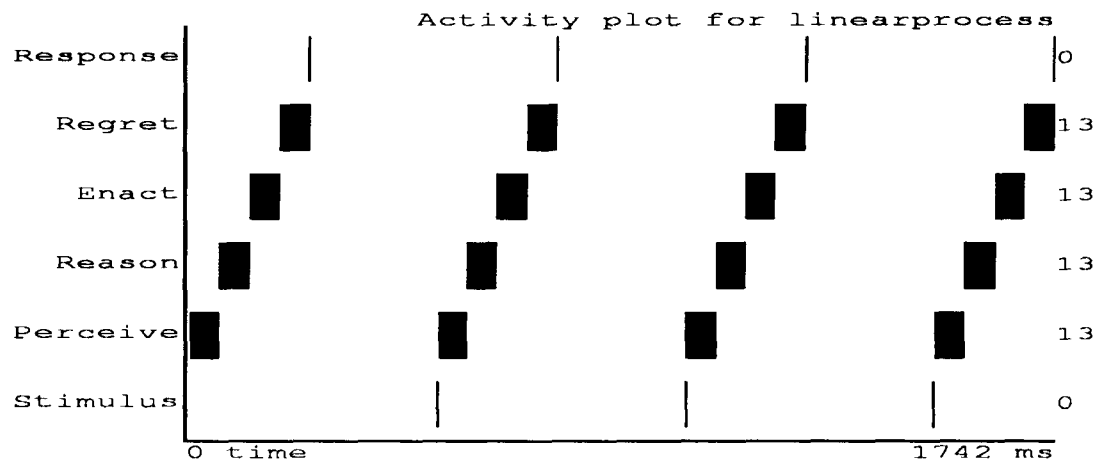


Figure 4: Activity chart for linear process

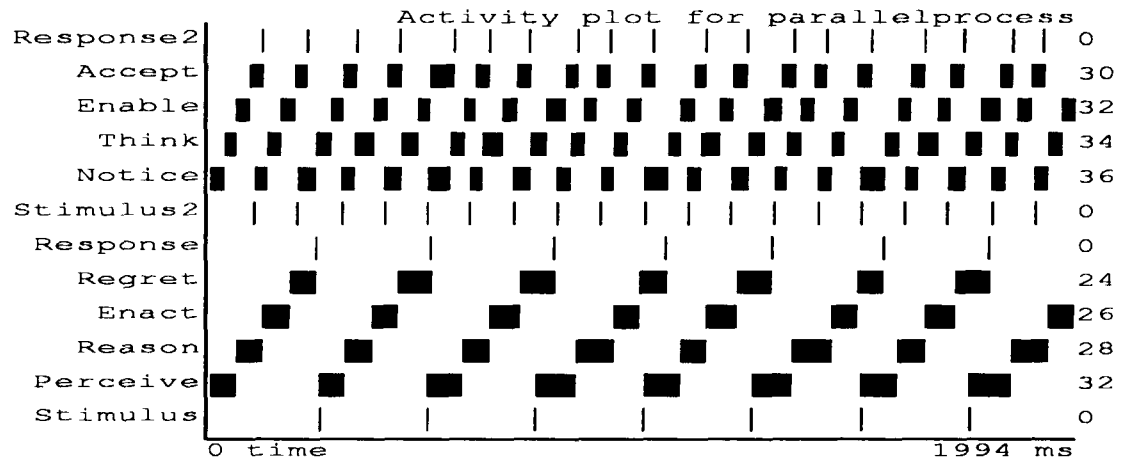


Figure 5: Activity chart for parallel process

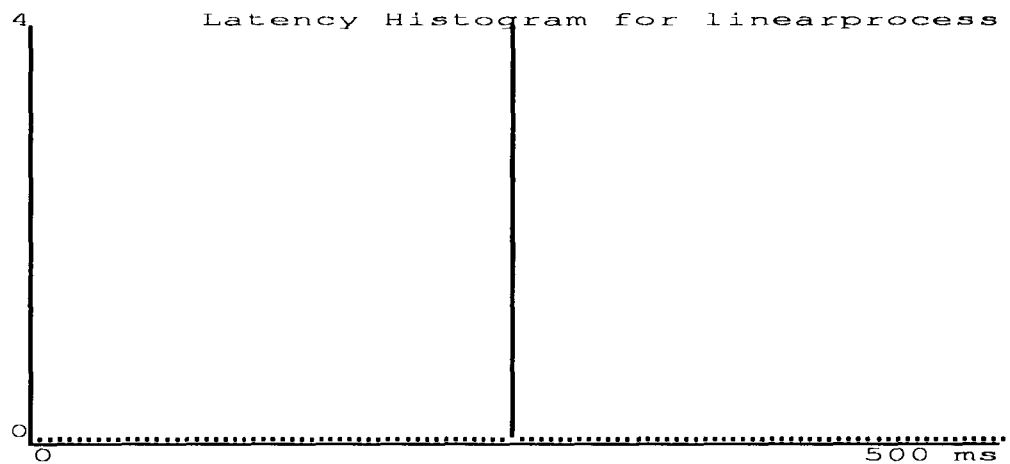


Figure 6: Latency chart for linear process

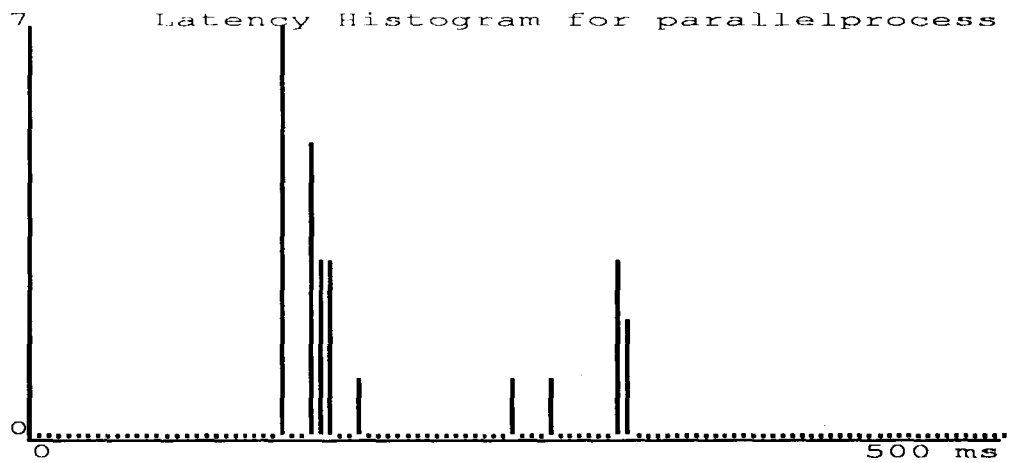


Figure 7: Latency chart for parallel process

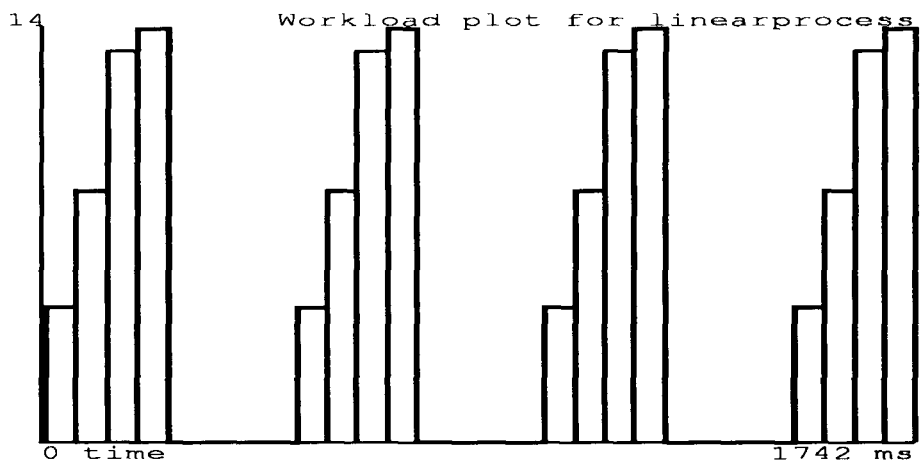


Figure 8: Workload chart for linear process

heights are. While we added only one competing task, we see more than a simple doubling of the workload value, which is as expected. Once the simulation results are calibrated to real human behavior requirements, the models can be used to evaluate scenarios where overload conditions can occur. One outcome of such studies could simply be coupling of previously unrelated processes, so that simultaneous demands might be avoided to some extent, thus reducing the synergistic effects of the multiple demands.

Difficulties with approach

One of the primary difficulties with human factors modeling when involved with the man machine interface is the wide dynamic range possible in the modeling. Human reactions are on the order of milliseconds with a wide variance in actual response times, whereas the performance of typical electronic systems are deterministically quantifiable in nanoseconds. It is not immediately clear how such widely varying systems should be integrated, and what type of results one should expect. One possible approach is to abstract the behavior of electronics system, so the fidelity of both the electronics and human systems are of an equivalent level. However, such models are not generally available.

Another problem arises with the obvious stochastic behavior of a human model. Monte Carlo simulations will be necessary to produce accurate and meaningful results, but are unfortunately typified by heavy execution requirements since so many trials will have to be run. Such runtime requirements when modeling large complex systems may prove costly.

Finally, we must note that VHDL is far from ideal for system level modeling where the system includes the human operator. Most notable of the restrictions are the lack of variant records and generic functions. The variant records force us to consume excessive amount of memory with our token and workload signals.

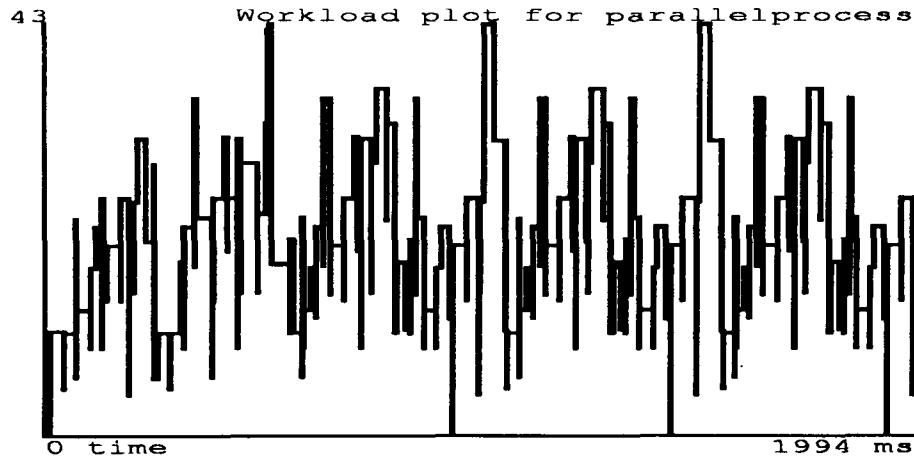


Figure 9: Workload chart for parallel process

The lack of generic functions has resulted in a fairly large expansion of the actual amount of code that is necessary to write and maintain, although the use of macros has somewhat alleviated this problem.

5 Future Goals

These generic human factors models and support packages have all been built and debugged using the Vantage Analysis tools. Future efforts will involve porting the graphics portion of the model to other tools. The VHDL code is generic, so we do not expect portability problems (outside of the potential for problems with the non-associative workload bus resolution function). Our primary future efforts will focus on establishing a library of configured components that reflect human behavior, as well as calibrating the simulation results against measured laboratory performance. Finally, it seems plausible that the full functional capabilities of VHDL can be utilized to develop detailed functional and decision making modules for representing more complex, data-dependent processes.

It is evident from this modest demonstration that VHDL has utility for modeling human factors. Even though VHDL was not explicitly designed to model such issues as global workload, it can easily represent task networks typically encountered in such operability modeling. It has the added benefit that these human factors models can be easily integrated with machine performance models, enabling powerful simulation and analysis very early in program development as well as through more detailed levels of design. In addition, straightforward generic output reporting modules rapidly generate workload assessment and reporting, which are the usual measures of interest in such studies. All of these models and the reporting capabilities were developed in approximately one person-week of labor, which speaks well for the utility and productivity of the particular VHDL development environment used here. An argument could be put forth that the ease of such modeling, and the continual development, enhancement, and acceleration of VHDL tools and capabilities makes this modeling environment an appropriate base for further research.

6 Acknowledgements

The token portion of the VHDL model was based upon work originally performed under the VHDL portion of the Graphics Processor Definition program funded by the Wright Laboratory Cockpit Integration Directorate (WL/KTD).

The particular human resource characterization used as an example here was developed under private research grants by Honeywell Incorporated, February 1989.

References

- [1] Honeywell, "Graphics Processor Description VHDL Methodology: Working Document," Honeywell, 1991.
- [2] Robert A. North and Victor A. Riley, "W/INDEX, A Predictive Model of Operator Workload," Honeywell, 1989.