

# Abstraction Networks for Modeling and Simulation

Maziar Khosravipour, Herbert Grünbacher  
Institut für Technische Informatik  
Technische Universität Wien  
Treitlstraße 3 - 1822  
A-1040 Vienna, Austria  
email: {maziar, gruenbacher}@vlsivie.tuwien.ac.at

## Abstract

*Any system is potentially subject to multiple objectives and no single model will be able to capture all the objectives of the system. To deal with multiple objectives, we have to create and manage multiple models of the system each covering certain aspects. Each aspect represents a level of abstraction for the system. We present the concept of abstraction networks, a formalism for multilevel hierarchical systems. Abstraction networks are constructed by applying abstraction mechanisms to existing models and organizing them hierarchically. They comprise various representations of a system at different levels of abstraction. The concept of abstraction networks and its extension by VHDL is applied to a FIFO. Further, a knowledge-based framework which supports the process of modeling and simulation based on abstraction networks is specified. An important rationale of the framework is to capture simulation related issues such as simulation efficiency and model complexity.*

## 1. Introduction

VHDL, the hardware description language mostly used for hardware modeling lacks formal semantics, which covers the full set of language constructs and is computable in reasonable time. Although formal methods like BDDs can be applied to verification of VHDL descriptions, these methods are highly time and space consuming. We believe that the pragmatic way to deal with the problem of validation and verification is to analyze a system by using more than one method of representation and integrating the results into one framework.

The concept of abstraction network was introduced by [2] primarily for process abstraction, but we are convinced that it could also serve as a framework for design of computer systems as a whole, and hardware design in particular.

The classical design methods based on conventional definition of abstraction hierarchy suffers from the fact, that the one-dimensional ordering of abstraction hierarchy does not represent the reality of design processes. For instance, in most cases, a fully specified system description is not available at the beginning of the design process. Furthermore, the specification itself is subject to modifications and evaluations. So, the top-down design approach, starting with a specification and concretizing the specification by means of design and technology details up to an implementation, is mostly not applicable.

On the other hand, the classical top-down methodology offers no means to integrate the “bottom-up” knowledge of the designer into the design process. This information builds an important part of the design process as a steering mechanism to select the “right” description, especially in hardware design.

The “abstraction network” approach places the abstraction levels in a multidimensional space and establishes a partial ordering over these levels. This approach is formalized in [3]. The abstraction network can be extended by VHDL implementations derived from the abstract representations. Although a CAD framework for abstraction networks does not exist yet, its architecture is already specified. This framework will be realized within an ESPRIT basic research project (FOCUS driven Hierarchical Simulation Technology for reactive embedded systems).

## 2. Example: FIFO

### 2.1. Description

The FIFO consists of a certain number of memory cells, each containing an object (e.g. a 32-bit word). The FIFO acts as a buffer between a fast source and a slow sink of objects. The objects are written to the FIFO by the source, for instance the CPU of the system, at a given rate. The objects are read from the FIFO by another component, for instance the main memory, at a given rate, which usually is less than the rate of the write process. The main function of FIFO is to buffer the objects and write them out on the read request of the memory unit in the right order, i.e. this is to forward the first data written to FIFO to the memory first.

### 2.2. Observation Model (DATA1, DATA2)

The starting point of the conceptualization of the FIFO is the observation of the data stream running through the system (Figure 1).

In this model, called DATA1, the data are observed as a stream beginning at time 0. The FIFO can be considered as a moving frame to observe the data stream. At each point in time, a certain amount of data (= depth of FIFO) is visible through the frame. Two pointers for read and write operations manage the frame placement over the data stream. Each write operation shifts the frame one object to the right. The read pointer marks the currently read object.

Another approach to observe the data stream is using a fixed frame for observation (DATA2). The frame can be viewed as buffer holding certain amount of objects and releasing them in a first-in-first-out order. The cells of the buffer are indexed and can be accessed by their address.

### 2.3. Formal Model of FIFO (FORM)

For a formal description of FIFO, we will use the approach introduced by [1]. It is a pure behavioral model capturing the time dependent properties of the system. A formal description of FIFO can be derived from the observation model DATA2. The FIFO consists of  $n$  memory cells, numbered  $0, \dots, (n-1)$ . Following events and conditions characterize the system:

write\_request.....CPU initiates a write to FIFO;  
 read\_request.....Main memory signals that it is prepared to read from FIFO;  
 write<sub>*i*</sub>.....Data is being written to memory cell *i* of FIFO;  
 read<sub>*i*</sub>.....Data is being read from memory cell *i* of FIFO;  
 fifo\_empty .....FIFO is empty (no reading from FIFO is possible);  
 fifo\_full.....FIFO is full (no further writing to the FIFO is possible).

The relevant relations for the implementation are shown in Table 1.

### 2.4. State Machine Model of FIFO (FSA1, FSA2 and FSA3)

The controller part of the FIFO can be defined as a finite state automaton. There are two alternatives to represent the FIFO controller as a finite state automaton. We distinguish between the FIFO controller and the FIFO-RAM, where the data are stored. The first implementation is shown in Figure 2.

The other implementation of the FIFO controller can be obtained by adding a new transition to the definition of the state machine. Adding the transition "shift" which will

$write_i \in \left[ \begin{array}{l} write\_request \mid count_2(write_i) = count_2(read_i) \\ \text{and} \\ \neg \exists j < i : count_2(write_j) = count_2(read_j) \end{array} \right]$	
$read_i \in \left[ read\_request \mid P_i = \max\{P_k \mid k = 1, \dots, n\} \text{ and } P_i \neq 0 \right]$	
$P_i^j = \begin{cases} 0 & , \text{if } count_2(read_j) = count_2(write_j) \\ lcount_1(write_i) \circ next_1(write_j) \circ last_2(read_j) & , \text{else} \\ -lcount_1(read_i) & \end{cases}$	$P_i = \sum_j P_i^j \text{ (Priority)}$
Liveness property: $TIME_1(read_i) = TIME_2(write_i) + \delta, \delta \geq 0$	
Safety property: $\forall i, j: TIME_1(write_i) \leq TIME_1(write_j) \Rightarrow TIME_1(read_i) \leq TIME_1(read_j)$	

Table 1 - Formal model of FIFO

take place after each read operation leads to a significant reduction of the number of the states. This implementation corresponds to a more complex implementation of the

FIFO-RAM (Figure 3). A third implementation of FI can be done using state diagrams which can be easily transformed to VHDL (Figure 4).

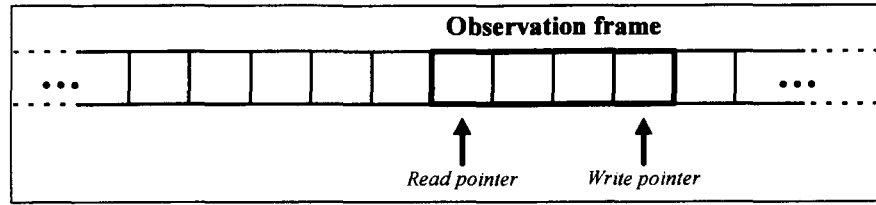


Figure 1 - Observation model for FIFO

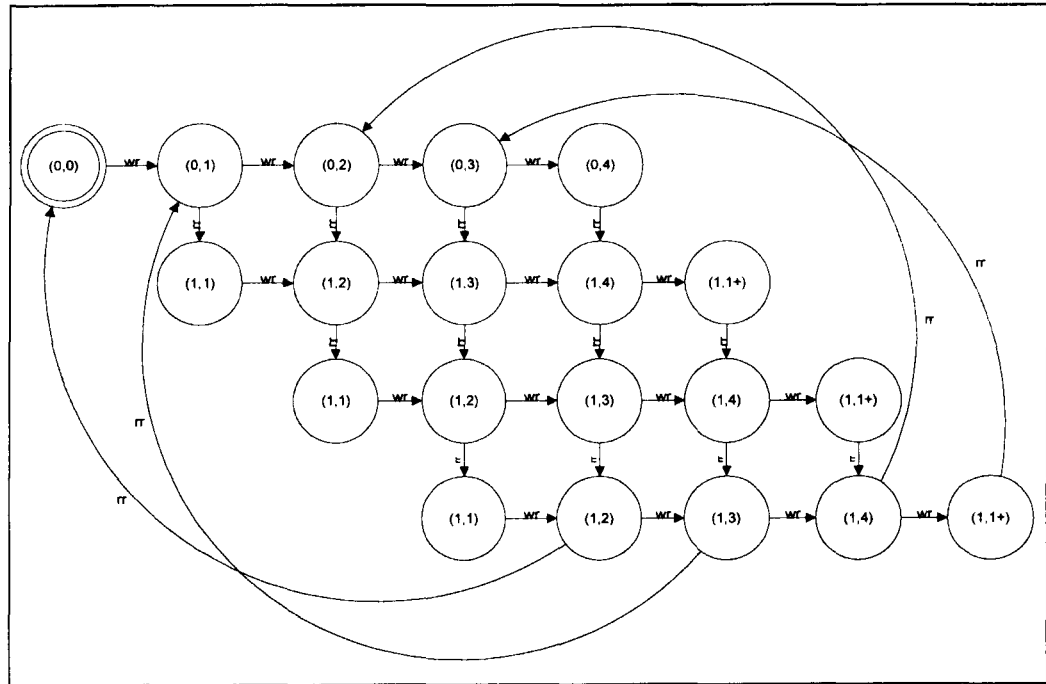


Figure 2 - Finite state automaton FSA1

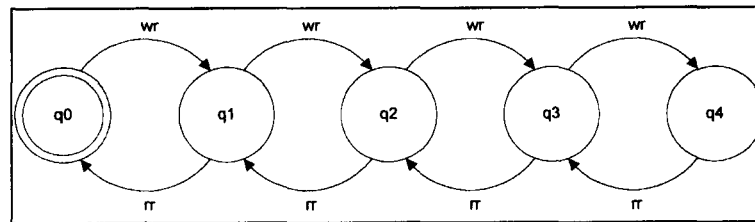
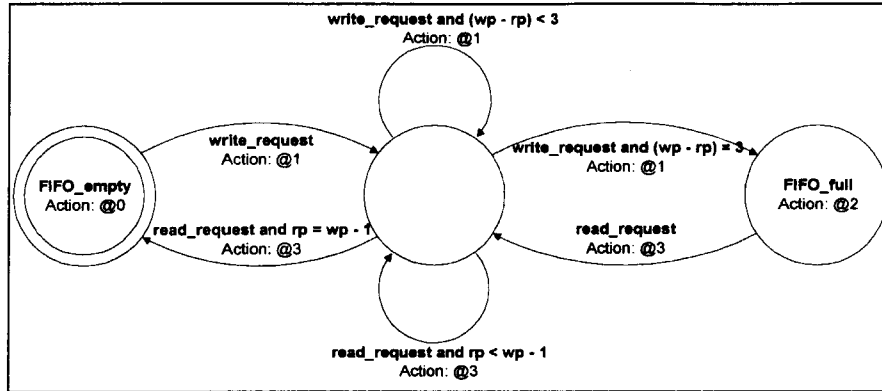


Figure 3 - Second FSA implementation FSA2



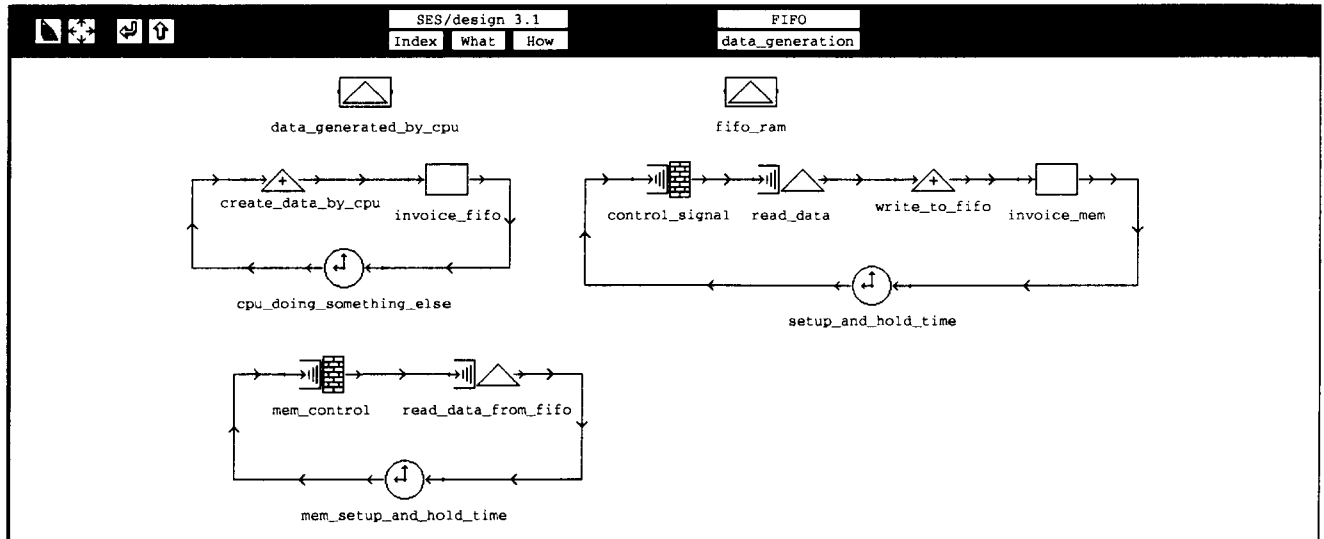
Action	Description
@0	wp := 0; rp := 0; fifo_empty := TRUE;
@1	fifo_ram(wp) := fifo_in; wp++; fifo_empty := FALSE;
@2	fifo_full := TRUE;
@3	fifo_out := fifo_ram(rp); rp++; fifo_full := FALSE;

**Figure 4 - State diagram model for FIFO**

### 2.5. Performance Model of FIFO (PERF)

A system level model of FIFO embedded in a given environment will provide appropriate means to analyze the performance and throughput of the FIFO. It also enables us to specify quantitatively the dimension of the component needed to meet the requirements of the whole system.

These requirements are given as probabilistic functions specifying the dynamic behavior of other components of the system. For our example, we model a computer system consisting of a data source (e.g. a CPU) and a data processing unit (e.g. a memory unit). The quantitative simulation of the system is done by ses/workbench™ (Figure 5).



**Figure 5 - Performance Model PERF of FIFO**

### 3. Abstraction Network for FIFO

The above representations of the FIFO and other models can be integrated into an abstraction network determining the abstraction relation among the descriptions. The abstraction network and the abstraction mechanisms for the FIFO are given in Figure 6.

FSA2 is shown to be an abstraction of FSA1. The abstraction mechanism used for the transformation of FSA1 to FSA2 is total system morphism (TSM). System morphism [6] is a behavior and structure preserving abstraction mechanism. Other abstraction mechanisms indicated in Figure 6 are Inductive inference method IND (DATA2  $\rightarrow$  FSA1) and abstraction by reduction RED (FSA1  $\rightarrow$  PERF, FSA2  $\rightarrow$  PERF). An extensive description of the abstraction mechanisms is given in [2].

These models can now answer various questions that relate to the proper level of abstraction. By applying the performance model PERF, we can dimension the FIFO for a given system throughput. The finite state automata FSA1, FSA2 and FSA3 offer alternatives to implement the FIFO controller in VHDL.

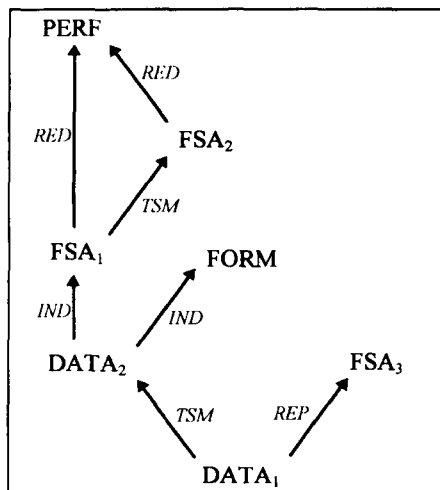


Figure 6 - Abstraction Network for FIFO

### 4. VHDL Implementation of FIFO

We will now extend the abstraction network for the 4-cell FIFO given in the last chapter by three VHDL descriptions derived directly from other representations of the system. For all three implementations, there is a common entity declaration, representing the FIFO and its interface (Code 1).

```
entity fifo is
port (write_request : in bit;
      read_request : in bit;
      fifo_in : in bit_vector (31 downto 0);
      fifo_empty : out bit := '1';
      fifo_full : out bit := '0';
      fifo_out : out bit_vector (31 downto 0)
);
end entity fifo;
```

Code 1 - Entity declaration of FIFO

#### 4.1. Implementation 1

The first implementation of FIFO can be obtained transforming the observation model of the FIFO into a VHDL description. This will be a very high level behavioral representation of the FIFO which does not address the implementation issues for a FIFO in any way. We consider two variables representing the write position and the read position in a data stream generated by a data source and to be processed (i.e. consumed) by a memory unit. The code fragment in Code 2 (Appendix) gives the VHDL description for this implementation.

#### 4.2. Implementation 2

This VHDL implementation is derived from the state machine description FSA2 of FIFO. The FIFO is implemented as an array containing four registers. To provide the function of FIFO (first-in first-out order data), the read operation is always performed on the first register. After a read operation, the contents of the FIFO are shifted toward the first cell. Write operations are done on the first free register.

#### 4.3. Implementation 3

A totally different approach to implement the FIFO is derived from the formal description of FIFO. As a side effect, this VHDL description fulfills the safety and liveness requirements as stated as a part of the formal description of FIFO. The selection of candidates for write operation (COUNT(write<sub>i</sub>) = COUNT(read<sub>i</sub>)) is implemented by a valid bit. A cell containing data not yet read is marked as valid and can therefore not be written. If all cells have the valid bit set, then the FIFO is full and write operations have to be postponed until some data is read.

The sequence of the data to be read is determined similarly to the formal description, by a priority structure implemented as log(fifo\_depth) bits appended to each

denoting its priority for the read operation. After a FIFO cell has become valid (write operation to the cell), the priority is incremented after each write operation to the FIFO. On a read request, the cell with the highest priority will be read by the memory and the priority is set back to zero. Code 4 (Appendix) shows the VHDL description of the FIFO.

## 5. A Framework Based on Abstraction Networks

The main idea of abstraction networks is to provide a formalism able to capture the multiple objectives of a system being designed or analyzed. For each set of questions, a corresponding model is constructed at the proper level of abstraction. The use of adequate models facilitates traversal across levels of abstraction. The focus of this approach is on the development of simulation models comprising characteristics of a given system. Models, at various levels of abstraction and granularity constitute a network allowing the „right“ model to be simulated for a specific purpose. In our framework, we provide expert systems-based techniques for

- 1) choosing the adequate model for simulation,
  - 2) applying admissible transformations to existing models resulting in new more abstract models to improve simulation efficiency, and
  - 3) cross-checking of model compliance with design specifications, constraints, and requirements.
- The framework is built around VHDL as core language.

### 5.1. Organization of the Framework

The concept of the blackboard architecture [4][5] has been adapted to match the requirements of our framework. The elements of the framework are shown in Figure 7. A detailed framework specification is given [3].

The framework consists of a knowledge base containing the models which represents the knowledge about the system to be designed respectively simulated; the system database is the blackboard where information about the system is collected and communication knowledge exchange among the models is performed; the inference machine can deduce new knowledge about the system by executing knowledge acquisition procedures and also generating new more abstract models by applying abstraction mechanisms to the existing models in the model base; human intervention is needed for adding new models to the model base, monitoring and steering the abstraction and deduction tasks performed by the

inference machine. The human expert has also to schedule proper reactions if model inconsistencies are detected.

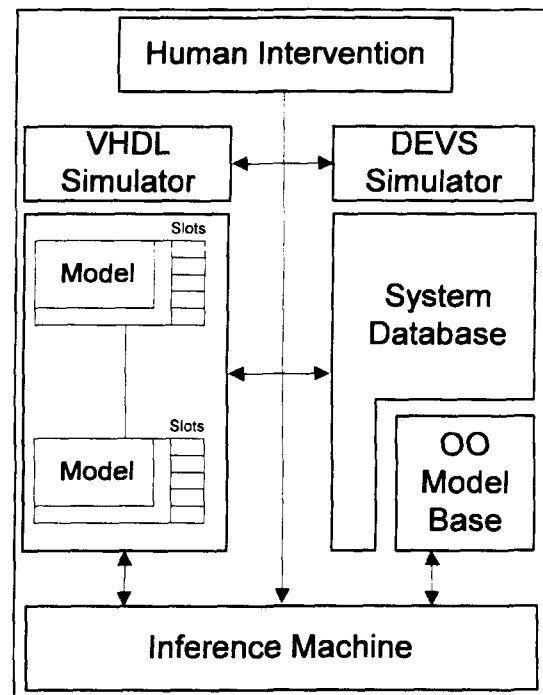


Figure 7 - Framework architecture

## 6. Summary and Conclusion

Applying the approach of abstraction networks to system modelling enables us to ensure the correctness and feasibility of systems to be developed starting with a specification, which might not cover all relevant aspects. Using abstraction networks, it is possible to integrate evolutionary development methods as known to software engineering into the hardware design process. Since formal semantics covering the full set of VHDL constructs are not available yet - it is however not granted that there will ever be one at all -, the only way to verify and validate a particular system implementation is to cross-check the implementation with other models of the system, which are known to reflect the behavior of the desired system correctly.

Abstraction networks can be extended by adding new arcs to the network denoting the refinement relation "is concretization of". Zeigler [6] discusses "justifying conditions" where a mapping may be produced from a higher level to a lower level of abstraction. Essentially, these conditions are constraints that permit the inference of structure from behavior in the hierarchy of description levels. In contrast to elimination of knowledge by

abstraction, the addition of knowledge about a model by concretization is not yet supported by general methods.

We suggest a framework based on the concept of abstraction networks for development of multimodels for hardware systems. The aim of the framework is primarily to improve simulation efficiency and knowledge deduction for model transformation. Its focus is on the integration of simulation models at various levels of abstraction developed by tools such as system level design tools, into a model base. It provides a representation of the characteristics of the system being modeled by deducing facts about the system from the model base. It should also facilitate transformation of system descriptions to higher abstraction levels in order to improve the simulation efficiency by simultaneously preserving the overall behavior of the system.

## 7. References

- [1] Alagar V.S., Ramanathan G.: "Functional Specification and Proof of Correctness for Time Dependent Behaviour of Reactive Systems", *Formal Aspects of Computing*, 3:253-283, 1991.
- [2] "Process Abstraction in Simulation Modeling," Fishwick, P.A., in *Artificial Intelligence, Simulation and Modeling*, Widman, L.E., Helman, D.H., Loparo, K.(eds); John Wiley and Sons; 1988.
- [3] "Modeling and Simulation of Computer Systems based on Abstraction Networks," Khosravipour M., Ph.D. Thesis, Vienna University of Technology; 1996.
- [4] "*Blackboard Systems*," Morgan T., Engelmores R. (eds); Addison-Wesely; 1988.
- [5] "Blackboard systems: The blackboard model for problem solving and the evolution of blackboard architectures," *AI Magazine*, 7, pp. 38-53, Summer 1986.
- [6] "Multifaceted Modeling and Discrete Event Simulation," Zeigler, B.P.; Academic Press; 1984.

## Appendix

```

architecture observe of fifo is
subtype fifo_word is bit_vector(31 downto 0);
type ram is array (0 to 3) of fifo_word;
signal fifo_ram : ram;
begin
  process
    variable write_pointer, read_pointer : integer := 0;
    begin
      wait for write_request, read_request;
      if (write_request'event and write_request = '1') then
        if (write_pointer - read_pointer) < 3 then
          fifo_ram(write_pointer) <= fifo_in;
          write_pointer := write_pointer + 1;
          fifo_empty <= '0';
        elsif (write_pointer - read_pointer) = 3 then
          fifo_ram(write_pointer) <= fifo_in;
          write_pointer := write_pointer + 1;
          fifo_full <= '1';
        else
          null;
        end if;
      end if;
      if (read_request'event and read_request = '1') then
        if (read_pointer < write_pointer - 1) then
          fifo_out <= fifo_ram(read_pointer);
          read_pointer := read_pointer + 1;
          fifo_full <= '0';
        elsif (read_pointer = write_pointer - 1) then
          fifo_out <= fifo_ram(read_pointer);
          read_pointer := 0;
          write_pointer := 0;
          fifo_empty <= '1';
        else
          null;
        end if;
      end if;
    end process;
end observe;

```

Code 2 - VHDL observation model for FIFO

```

architecture FSA2 of fifo is
...
process
variable fifo_index : integer := 0;
begin
wait for write_request, read_request;
if (write_request'event and write_request = '1') then
if (fifo_index < 3) then
fifo_ram(fifo_index) <= fifo_in;
fifo_index := fifo_index + 1;
fifo_empty <= '0';
elsif (fifo_index = 3) then
fifo_ram(fifo_index) <= fifo_in;
fifo_index := fifo_index + 1;
fifo_full <= '1';
else
null;
end if;
end if;
if (read_request'event and read_request = '1') then
if (fifo_index > 0) then
fifo_out <= fifo_ram(0);
for i in 0 to 2 do
fifo_ram(i) := fifo_ram(i+1);
end for;
fifo_index := fifo_index - 1;
if (fifo_index = 0) then
fifo_empty <= '1';
end if;
else
null;
end if;
end if;
end process;
end FSA2;

```

Code 3 - FIFO description based on FSA2

```

architecture formal of fifo is
subtype priority is integer range 0 to 3;
subtype fifo_word is bit_vector(31 downto 0);
type fifo_cell is record
cell : fifo_word;
valid : bit;
c_p : priority;
end record;
type ram is array (0 to 3) of fifo_cell;
begin
process
variable t_pointer : integer;
variable fifo4 : ram;
begin
wait for write_request, read_request;
if (write_request'event and write_request = '1') then
for i in fifo4'range loop
if (fifo4(i).valid = '0') then
fifo4(i).cell := fifo_in;
fifo4(i).valid := '1';
for j in fifo4'range loop
if j <> i then
if fifo4(j).valid = '1' then
fifo4(j).c_p := fifo4(j).c_p + 1;
end if;
end if;
end if;
end if;
end for;
end if;
if (read_request'event and read_request = '1') then
t_pointer := 0;
for i in 1 to fifo4'high loop
if (fifo4(i).c_p > fifo4(t_pointer).c_p) then
t_pointer := i;
end if;
end for;
if (fifo4(t_pointer).c_p <> 0) then
fifo_out <= fifo4(t_pointer).cell;
fifo4(t_pointer).valid := '0';
fifo4(t_pointer).c_p := 0;
end if;
end if;
-- set fifo_empty and fifo_full
end process;
end formal;

```

Code 4 - FIFO description based on FORM