Proiectarea Sistemelor Digitale cu Instrumente HDL

-LINKED LIST-

(Liste Înlănțuite)

Conf. Dr. Ing.: Botond KIREI

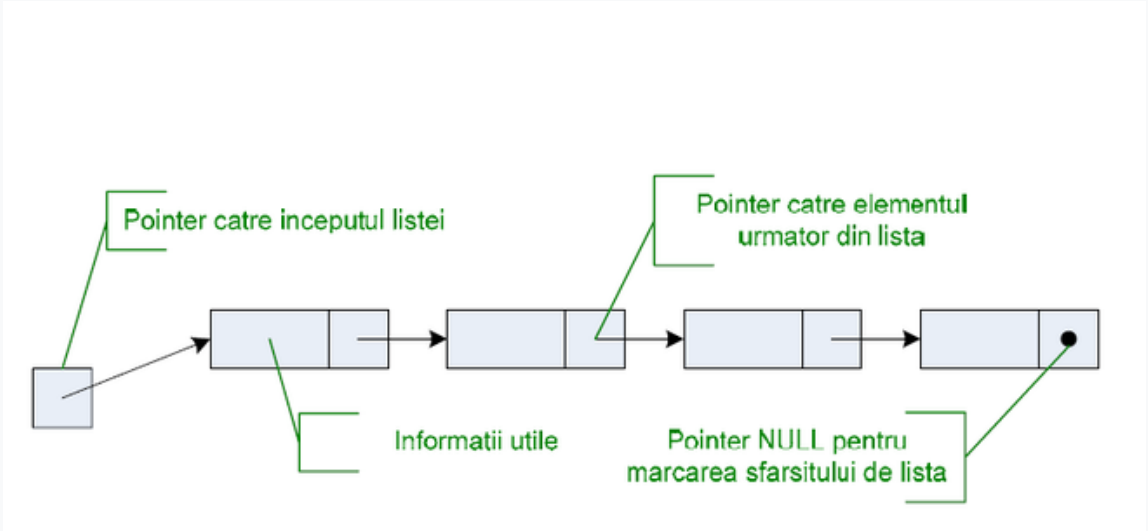
Student:

Bianca Bota

Alocarea dinamică a memoriei permite utilizatorului să realizeze spațiu în timpul rulării, mai degrabă decât în timpul elaborării (în VHDL, acest lucru se poate face folosind specificatorul *ACCESS*). Cea mai semnificativă decizie pentru programator este modul de organizare a structurilor de memorie așa cum este solicitat. Memoria stației de lucru alocată, de obicei, nu este un singur bloc contigu. Există mai multe metode cunoscute pentru organizarea structurilor de date [1], inclusiv listele înlănțuite.

Listele simplu înlănțuite sunt structuri de date dinamice, omogene. Listele sunt alocate ca elemente separate de memorie, ci nu ca blocuri omogene de memorie. Fiecare nod al listei conține, în afara de informația utilă, adresa următorului element. Această organizare permite numai acces secvențial la elementele listei. Pentru accesarea listei trebuie cunoscută adresa primului element (numită capul listei); elementele următoare sunt accesate parcurgând lista.

Lista simplu înlănțuită poate fi reprezentată grafic astfel:



[1]

O listă înlănțuită (linked list) poate fi utilizată atunci când numărul total de elemente nu este cunoscut în avans.
O astfel de listă este un șir sau un record, fiecare dintre acestea conținând o valoare (dată) și un pointer către următorul *item* din listă. Se pot folosi tipuri incomplete pentru a lucra cu liste înlănțuite în VHDL. [1]

De obicei, listele înlănțuite sunt implementate cel mai convenabil folosind clase într-un limbaj de programare orientat pe obiecte, precum C++, VHDL. VHDL are câteva caracteristici orientate pe obiect care pot fi utilizate pentru a elimina sau a reduce complexitatea implementării.

1. Implementarea unei liste

Implementarea unei liste utilizând limbajul hardware de programare VHDL, presupune parcurgerea următorilor pași: []

1.1 Crearea unui package VHDL

Pachetul trebuie să cuprindă întreg codul necesar implementării.
Un pachet VHDL este o colecție de tipuri, obiecte sau subprograme care pot fi importate într-un alt fișier VHDL. Majoritatea modulelor VHDL importă pachetul *std_logic_1164* din biblioteca IEEE.

[1] Advanced VHDL Constructs And Methodologies For Design Verification, Subbu Meiyappan, James Steele

```
package DataStructures is
   -- Public declarations go here
end package DataStructures;

package body DataStructures is
   -- Private declarations and implementations go here
end package body DataStructures;
```

Un pachet este format din două părți; o regiune declarativă și un corp. Regiunea declarativă este locul în care se regăsețecodul ce ar trebui să fie vizibil pentru toți utilizatorii pachetului. Corpul este privat și nu este accesibil din exterior. [2]

- ***Tipul PROTECTED***

  Tipurile *PROTECTED* sunt construcții asemănătoare clasei în VHDL. Pot conține subprograme, declarații (de tip) și variabile. Un tip *PROTECTED* constă dintr-o declarație publică și un body privat.

```
package DataStructures is

   type LinkedList is protected

      -- Prototypes of subprograms
      procedure Push(constant Data : in integer);
      impure function Pop return integer;
      impure function IsEmpty return boolean;

   end protected;

end package DataStructures;

package body DataStructures is

   type LinkedList is protected body
   end protected body;

end package body DataStructures;
```

Tipul *LinkedList* este protejat (*protected*), deoarece va conține tot ce este legat de lista înlănțuită (*linked liste*). O adăugare a unui alt tip de structură de date în pachet, presupune existența unui alt tip protejat (*protected*) .

În cadrul regiunii declarative de tip protejat (*protected*), se regăsec trei subprograme, momentan fără implementare. Pentru ca subprogramele să fie vizibile în afara tipului protejat (*protected*), acestea trebuie declarate.
Cele trei subprograme sunt metodele standard de listă ale unei liste înlănțuite: *Push, Pop* și *IsEmpty*. [3]
Push adaugă un element nou la începutul listei.
Pop elimină un element din finalul listei.
IsEmpty este o funcție care returnează true (adevărat) dacă lista este goală.

- ***Record***

Record este un tip compozit care poate conține mai multe diferite tipuri de membri multpli. Funcționează asemănător cu o structură din limbajul de programare C. Când un semnal sau o variabilă este creată din tipul *Record*, membrii datelor pot fi referiți folosind notația „ . ”. De exemplu *MyItem.Data*.

```
type LinkedList is protected body
```

2 HOW TO CREATE A LINKED LIST IN VHDL, https://vhdlwhiz.com/linked-list/

3 Creating efficient memory models in VHDL, Microelectronics Joumal, 25 (1994) 6 John Wilson, Vantage Analysis Systems (Europe)

```
   type Item is record
      Data : integer;
      NextItem : Ptr;
   end record;

end protected body;
```

Tipul *Record* este declarat în corpul tipului *Protected*. Regiunea declarativă a unui tip *Protected* nu poate conține alte declarații de tip sau semnal, prin urmare necesită declarate în corpul de tip *Protected*.

- ***ACCESS TYPE***

Tipurile *Access* reprezintă pointeri VHDL. Folosindu-le, putem crea în mod dinamic obiecte în timpul rulării.

```
package body DataStructures is

   type LinkedList is protected body

      -- Declaration of incomplete Item type
      type Item;

      -- Declaration of pointer type to the Item type
      type Ptr is access Item;

      -- Full declaration of the Item type
      type Item is record
         Data : integer;
         NextItem : Ptr; -- Pointer to the next Item
      end record;

      -- Root of the linked list
      variable Root : Ptr;

end package body DataStructures;
```

=> un nou tip de acces numit *Ptr* care va indica un obiect creat dinamic de tipul *Item*. [1]

Un nod de listă înlănțuită trebuie să conțină o referință către următorul nod

din listă. Acest lucru este implementat în înregistrarea/declararea *Item* -ului cu ajutorul membrul *NextItem*. *NextItem* este de tip *Ptr*, care la rândul său este un indicator înapoi la tipul *Item*. Pentru a evita această problemă, se declară întâi *Item* –ul ca un tip incomplet. Apoi, se declară tipul *Ptr* ca un pointer pentru tipul incomplet. În cele din urmă, se specifică declarația completă a tipului *Item*. Ultimul lucru făcut a fost declararea unei variabilă numită *Root* de tip *Ptr*. Aceasta va fi rădăcina listei înlănțuite. Dacă lista este goală, valoarea *Root* va fi nulă. În caz contrar, va indica către primul element din listă.

2 METODELE LISTEI ÎNLĂNȚUITE

1 PUSH

Push este singurul subprogram de tip procedură. Funcțiile din VHDL necesită o valoare de returnat (*return*) care nu poate fi omisă. Pentru a evita folosirea unei valori de retur fictivă, este de preferat folosirea unei proceduri, și anume Procedura *Push.* [2]

```
procedure Push(Data : in integer) is

   variable NewItem : Ptr;
   variable Node : Ptr;
begin
   -- Dynamically allocate a new item
   NewItem := new Item;
   NewItem.Data := Data;

   -- If the list is empty, this becomes the root node
   if Root = null then
      Root := NewItem;

   else
      -- Start searching from the root node
      Node := Root;

      -- Find the last node
      while Node.NextItem /= null loop
         Node := Node.NextItem;
      end loop;
```

```
      -- Append the new item at the end of the list
      Node.NextItem := NewItem;
   end if;
end;
```

Primul lucru care se întâmplă este faptul că un nou obiect de tip *Item* este alocat dinamic. Acest lucru se realizează folosind noul cuvânt cheie. De fiecare dată când se utilizează noul cuvânt cheie, memoria dinamică este consumată de către simulator.
Restul codului este doar un algoritm de listă înlănțuită standard. Noul *Item* este anexat la sfârșitul listei sau devine elementul *root* dacă lista este goală.

2 POP

Pop elimină cel mai vechi element din listă și îl returnează apelantului. Dacă se elimină doar referința la elementul popped și se returnează, va exista o anumită pierdere de memorie în simulator. După ce apelul funcției se termină, referința la obiectul creat dinamic se pierde pentru totdeauna.
Implementare funcție *POP*:

```
impure function Pop return integer is
   variable Node : Ptr;
   variable RetVal : integer;
begin
   Node := Root;
   Root := Root.NextItem;

   -- Copy and free the dynamic object before returning
   RetVal := Node.Data;
   deallocate(Node);

   return RetVal;
end;
```

Nu există garbage collection în VHDL înainte de VHDL, Iar VHDL-2017 nu este acceptat de majoritatea simulatoarelor. Prin urmare, este necesară apelarea *deallocate* înainte de a reveni din funcție.

Operatorul *deallocate* eliberează memoria folosită de un obiect alocat dinamic. Pentru fiecare apel către *new*, ar trebui să existe un apel *deallocate* înainte de a șterge referința la un obiect.

După apelarea *deallocate*, nu se poate face referire la datele la care se referă variabila *Nod*, soluția este copierea datele într-o variabilă locală înainte de *deallocate*, apoi returnarea valorii variabilei.

3 ISEMPTY

Verificarea dacă lista este goală sau nu, poate fi realizată pur și simplu verificând dacă nodul rădăcină (*root*) indică altceva decât nul:

```
impure function IsEmpty return boolean is
begin
   return Root = null;
end;
```

3 Operații cu liste

3.1 Parcurgere si afișare listă

Lista este parcursă pornind de la pointerul spre primul element si avansând folosind pointerii din structura pana la sfarsitul listei (pointer *NULL*).

3.2 Inserare element

Inserarea unui element se poate face la începutul sau la sfârșitul listei.

3.3 Căutare element

Căutarea unui element dintr-o listă presupune parcurgerea listei pentru identificarea nodului în funcție de un criteriu. Cele mai uzuale criterii sunt cele legate de pozitia în cadrul listei și de informațiile utile continute de nod. Rezultatul operației este adresa primului element găsit sau *NULL*.