Vectori de instanțe și structura `generate`

Oprițoiu Flavius
flavius.opritoiu@cs.upt.ro

4 noiembrie 2024

Introducere

Obiective:

- Construirea vectorilor de instanțe și configurarea blocurilor `generate`

De citit:

- "Advanced Module Instantiation", note de laborator [AMI**]

Vectorii de instanțe permit scrierea rapidă a mai multor instanțe ale aceluiași modul.

Blocurile generate oferă o variantă flexibilă de creare a unui număr mare de instanțe, cu interconexiuni complex.

Vectori de instanțe

Permite crearea rapida a mai multe instanțe ale aceluiași modul, atunci când toate instanțele sunt conectate la aceleași semnale. Utilizarea lor pentru proiecte cu interconexiuni complexe poate deveni dificilă.

Implementarea Verilog de mai jos construiește un convertor BCD8421 la E3 pentru numere cu k cifre, folosind k instanțe ale unui sumator pe 4 biți, numit *add4b*.

```
1  module bcde3conv #(
2    parameter k = 4          //number of digits
3  )(
4    input [4*k-1:0] bcd,     //bcd input number
5    output [4*k-1:0] e3      //e3 output number
6  );

8    add4b cnv[k-1:0] (.x(bcd),.y(4'd3),.z(e3));
9  endmodule
```

În linia 8 sunt construite k instanțe *add4b* folosind vectori de instante.

Vectori de instanțe (contin.)

Formatul vectorilor de instanțe:

```
module-name instance_name [top-index:bottom-index]
    (.p(s), ...)
```

Dacă lățimea semnalului *s* este egală cu numărul de instanțe înmulțit cu lățimea portului *p*, atunci *s* este partiționat egal în numărul de biți ai portului *p*, fiecare partiție fiind legată la una din instanțe create. Pentru exemplul *bcde3conv*, intrarea *bcd* va fi partiționată în grupe de 4 biți, fiecare grup fiind legat la una din cele *k* instanțe (similar pentru ieșirea *e3*).

Dacă lățimea semnalului *s* este egală cu laținea portului *p*, atunci întreg semnalul *s* este legat la toate instanțele. Pentru exemplul *bcde3conv*, valoarea biasului, 3, reprezentată pe 4 biți, ce se va aduna la fiecare cifră BCD8421, are aceeași lățime cu portul *y* al sumatrului și, deci, va fi conectată la toate cele *k* instanțe.

Blocuri generate

Reprezintă un mecanism mai flexibil de creare, într-un modul, a instanțelor multiple ale unui obiect. Pot fi generate următoarele tipuri de obiecte:

- unul sau mai multe module
- oricâte blocuri `initial`/`always`
- unul sau mai multe atriburi continue
- oricâte declarații de semnale

Instanțele generate sunt construite programatic în interiorul unui *bloc generate*, delimitat de cuvintele rezervate `generate` și `endgenerate`. *Blocul generate* utilizează bucla `for` pentru controlarea creării instanțelor. Variabila index a buclei `for` din *blocul generate* trebuie declarată de tipul `genvar` (valoare întreagă ne-negativă).

Pentru controlul mai detaliat asupra instanțierii se pot utiliza instrucțiuni `if ... else` și `case` în interiorul *blocurilor generate*.

Blocuri generate (contin.)

Bucla `for` din interiorul *blocului generate*:

- folosește o variabilă de tip `genvar` ca index
- are conținutul cuprins într-un bloc `begin ... end` *cu nume*

Convertorul anterior din BCD8421 la E3 este rescris ca mai jos:

```
1  module bcde3conv #(
2    parameter k = 4
3  )(
4    input [4*k-1:0] bcd,
5    output [4*k-1:0] e3
6  );
7    generate
8      genvar i;
9      for (i=0; i < k; i=i+1) begin: vect
10       add4b uconv(
11         .x(bcd[i*4+3:i*4]),.y(4'd3),.z(e3[i*4+3:i*4])
12       );
13     end
14   endgenerate
15 endmodule
```

Blocuri generate (contin.)

Blocul `begin ... end` începe în linia 9, unde `begin` este urmat de un identificator (în acest caz *vect*, dar se poate folosi orice identificator Verilog valid).

În blocul cu nume este construită o singură instanță, numită *uconv*, în fiecare iterație.

Asocierea porturilor utilizează indexul *i* pentru gruparea a 4 biți consecutivi din intrarea *bcd* și a 4 biți consecutivi din ieșirea *e3*. Grupele de 4 biți sunt realizate folosind expresia part-select $[i * 4 + 3 : i * 4]$.

La portul *y* al sumatoarelor este conectată valoarea 3, reprezentată pe 4 bits.

Problemă rezolvată

Unitatea de preprocesare a intrării a unei aplicații criptografice

Exercițiu: Să se construiască calea de date pentru unitatea de preprocesare a intrării (IPU, sau *unitatea*) a unui design Secure Hash Algorithm 2 (SHA-2) pe 256 de biți (vezi [FIPS15], secțiunea 5.1.1).

Soluție: Unitatea primește mesajul la intrare, îl extinde (*padding*) și îl furnizează la ieșire. La intrare mesajul este primit în pachete de 64 biți. La ieșire, mesajul extins este livrat în blocuri de 512 biți.

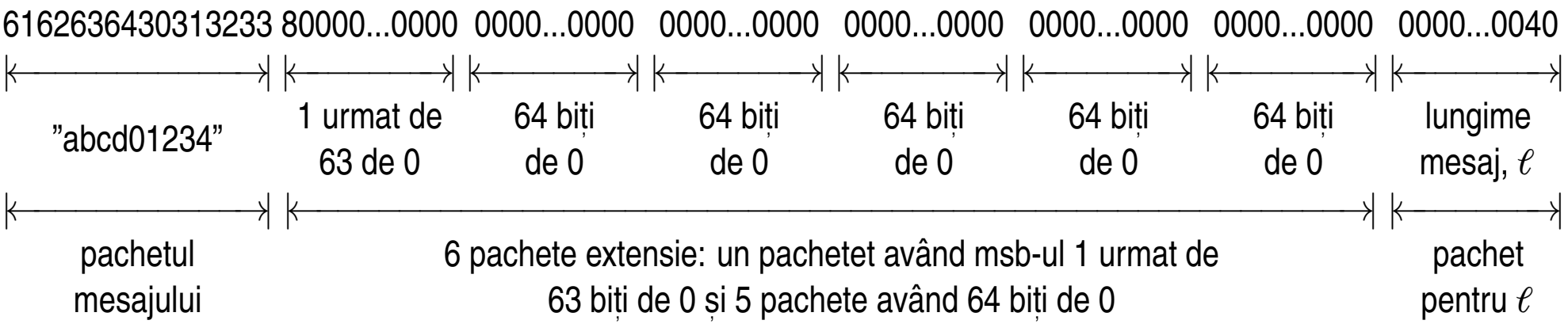
Extinderea mesajului: Condeirând un mesaj de ℓ biți, extinderea adaugă, în ordine:

- un bit de 1
- k biți de 0, cu $\ell + 1 + k \equiv 448 \pmod{512}$
- valoarea lui ℓ reprezentată pe 64 biți

Problemă rezolvată (contin.)

Secvența de pași exemplu pentru faza de preprocesare

Considerăm că unitatea primește mesajul ASCII "abcd0123" ($\ell = 64$ biți). Mesajului i se atașază un bit de 1 urmat de $k = 383$ biți de 0, urmați de valoarea 64 reprezentată pe 64 biți. Blocul de 512-biți de la ieșire este prezentat mai jos (cifre în hexazecimal):



Pentru mesajul de 72 de caractere ASCII: "Dear All, I am writing to give you an update on your submitted proposal.", unitatea va livra 2 blocuri de 512-biți

Problemă rezolvată (contin.)

Proiectarea căii de date a unității

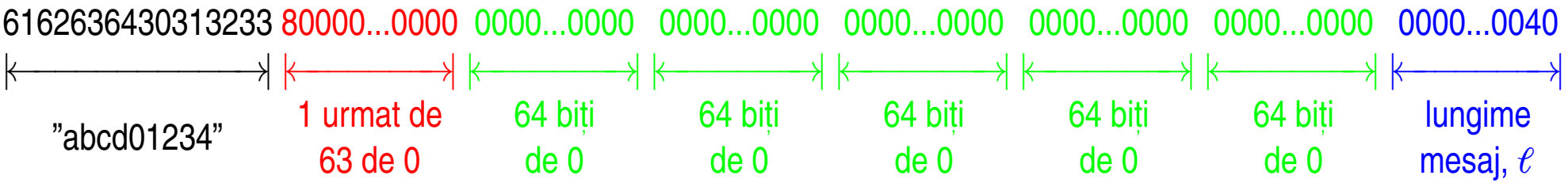
Cât timp mesajul nu a fost recepționat complet, în fiecare ciclu de ceas unitatea primește un pachet de 64-biți nou; $\Rightarrow$ 8 astfel de pachete formează un bloc de 512-biți. Aceste 8 pachete sunt stocate într-un register file, *regfl*, cu 8 registre a câte 64-biți.

Pentru că lungimea mesajului extins este multiplu de 512 iar ℓ este multiplu de 64: $\Rightarrow$ lungimea datelor extensie (bitul de 1, k biți de 0, ℓ pe 64 biți) este și ea multiplu de 64. Urmarea este că datele extensie pot fi împărțite în pachete de 64-biți și stocate în *regfl*.

Problemă rezolvată (contin.)

Proiectarea căii de date a unității

Pentru mesajul "abcd0123", împărțirea datelor extensie în pachete de 64-biți este ilustrata mai jos:



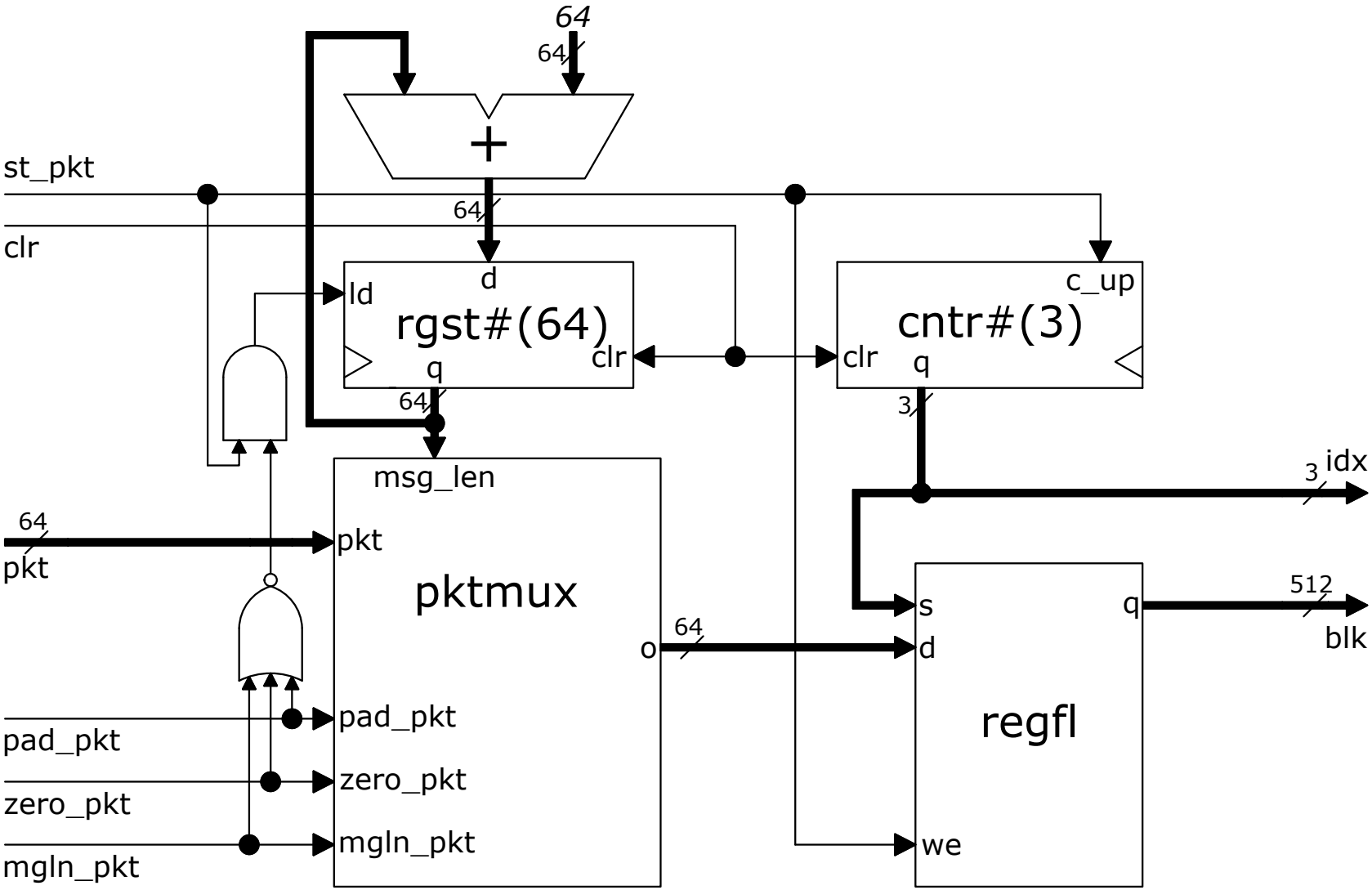
Astfel, există 4 tipuri de pachete prelucrate de calea de date:

1. pachet(e) mesaj
2. pachet extensie: având un bit de 1 urmat de 63-biți de 0
3. pachet(e) zero: având 64-biți de 0
4. pachet *lungime mesaj*: valoarea ℓ pe 64-biți

Lungimea mesajului, ℓ, este calculată în unitate folosind un registru pe 64-biți, incrementat cu 64 de unități de fiecare dată când un pachet este stocat în *regfl*.

Problemă rezolvată (contin.)

Proiectarea căii de date a unității



modulul *rgst* este disponibil ▸ aici, iar modulul *cntr* ▸ aici

Problemă rezolvată (contin.)

Proiectarea căii de date a unității

Calea de date are urmtoarele intrări, ilustrate în slide-ul anterior:

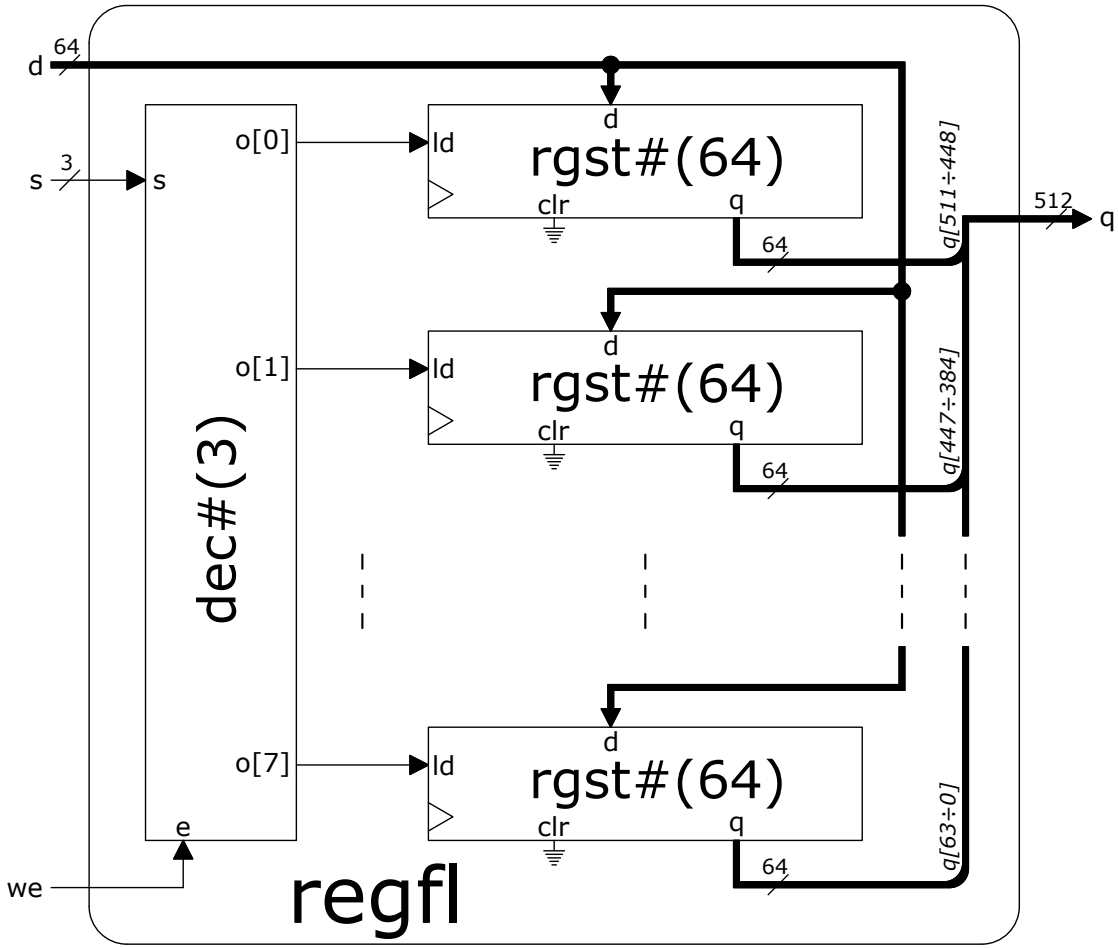
1. *pkt*: la care sunt primite pachete mesaj
2. *st_pkt*: activează stocarea pachetului curent
3. *clr*: resetează contorul și registrul pe 64-biți asociat lui ℓ
4. *pad_pkt*: activă dacă pachetul curent este de tip extensie
5. *zero_pkt*: activă dacă pachetul curent este de tip zero
6. *mgln_pkt*: activă dacă pachetul curent are lungimea mesajului, ℓ

Calea de date are următoarele ieșiri:

1. *idx*: următoare adresă disponibilă în *regfl*; indică, de asemenea, câte pachete din blocul curent au fost deja stocate
2. *blk*: blocul de 512-biți de ieșire

Problemă rezolvată (contin.)

Proiectarea register file-ului



dec#(3) este o instanță a modulului *dec* disponibil ▸ aici, parametrizată cu valoarea 3 pentru lățimea intrării de selecție.

Problemă rezolvată (contin.)

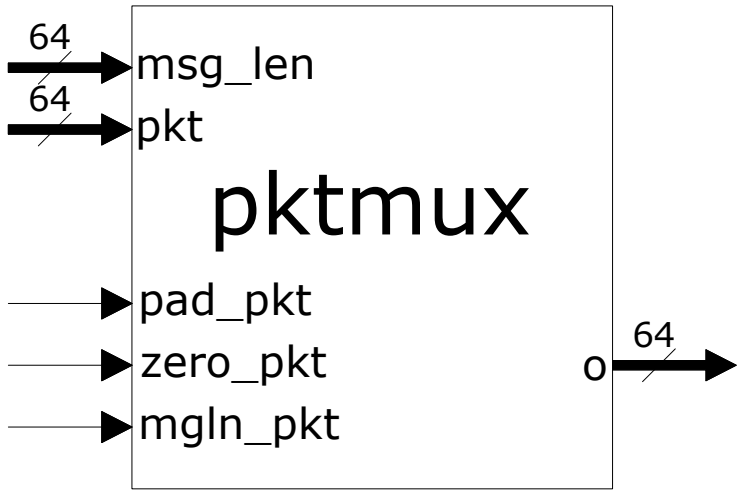
Proiectarea register file-ului

Register file-ul *regfl* asamblează 8 pachete consecutive primite la intrarea *d*, într-un block complet. Următoare adresă liberă în *regfl* este furnizată de un contor pe 3 biți la intrarea *s* iar stocarea pachetelor este activată de intrarea de enable *we*.

Ieșirea *q* a *regfl*-ului este construită concatenând conținutul tuturor registrelor interne, registrul de la adresa 0 furnizând cei mai semnificativi biți iar registrul de la adresa 7 pe cei mai puțin semnificativi.

Problemă rezolvată (contin.)

Multiplexorul de pachete



"Multiplexorul" de pachete, *pktmux*, furnizează pachetul curent de stocat în register file. Fiind 4 tipuri de pachete (vezi slide-ul 11), *pktmux* are 3 intrări de control, mutual exclusive:

1. *pad_pkt*: *pktmux* va livra un pachet extensie la ieșire
2. *zero_pkt*: va livra un pachet zero
3. *mgln_pkt*: va livra pachetul cu lungimea mesajului, furnizată de registrul pe 64-biți la intrarea *ms_len* a multiplexorului

Dacă niciuna din cele 3 linii de control nu sunt active, *pktmux* va livra un pachet mesaj, primit la intrarea *pkt*.

Referințe bibliografice

[AMI**] Advanced Module Instantiation. [Online]. Available: http://www.eecs.umich.edu/courses/eecs470/OLD/w14/labs/lab6_ex/AMI.pdf (Last accessed 17/04/2016).

[FIPS15] National Institute of Standards and Technology, "FIPS PUB 180-4: Secure Hash Standard," Gaithersburg, MD 20899-8900, USA, Tech. Rep., Aug. 2015. [Online]. Available: http://dx.doi.org/10.6028/NIST.FIPS.180-4 (Last accessed 06/04/2016).