VISUAL 8b v1.0

Visualizzatore **I**nterattivo **S**imulatore **U**nità **A**ritmetico **L**ogica a **8 b**it

Note di rilascio

1. Funzionamento generale e note per futura modifica

VISUAL 8b è stato realizzato interamente in Java.
Esso si basa su un sistema molto semplice: ad ogni stato della ALU ad 8 bit e di quella ad 1 bit è associato un insieme di immagini. Compito del programma è quello di stabilire per ogni funzione e per ogni input, quali immagini usare e quando visualizzarle.
Sebbene il funzionamento sia molto elementare la sua implementazione risulta piuttosto articolata, soprattutto per quanto riguarda la gestione delle immagini: essendo presenti infatti ben 16 funzioni calcolabili dalla ALU, è stato necessario creare un gran numero di immagini per rappresentare tutti i possibili stati della ALU a 1 bit e di quella a 8 bit (in totale 500 immagini png).

VISUAL 8b è stato realizzato usando l'IDE NetBeans v6.9. Essendo la grafica stata gestita in maniera visuale anziché da codice (scelta necessaria come si vedrà al punto 3), il codice relativo ad essa è stato creato in automatico dall'IDE. Dato che il programma è fortemente legato alla sua componente grafica sotto ogni aspetto, si consiglia fortemente per qualsiasi modifica di usare direttamente il progetto di NetBeans anziché il solo codice sorgente.

2. Il sistema delle immagini

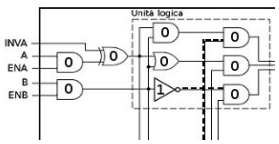
2.0 L'organizzazione

Le immagini sono divise in 2 classi: quelle della ALU a 1 bit e quelle della ALU a 8 bit.

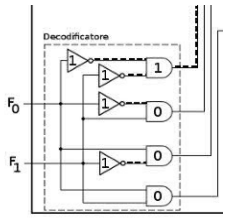
Dato che all'interno della ALU a 1 bit molte funzioni sono simili (basti pensare che il decodificatore con input 1,1 viene usato in 9 delle 16 funzioni), si è scelto anziché di rappresentare ogni singolo frame dell'animazione con unica immagine della ALU ad 1 bit, di scomporre l'immagine in 5 parti, in modo da poter usare la singola parte in più funzioni.

Le parti sono le seguenti:

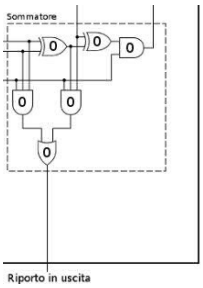
Parte logica



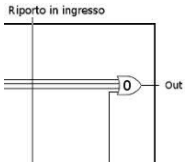
Decodificatore



Sommatore



Output



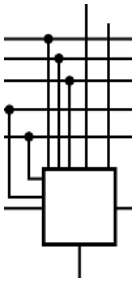
Parte di “raccordo”

Quindi ogni singolo stato della ALU a 1 bit è definito da una quintupla di immagini.

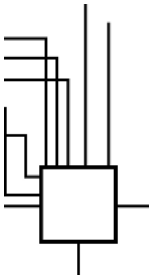
Per quanto riguarda l'immagine della ALU a 8 bit si hanno 3 differenti immagini:

head

standard



tail



L'immagine è quindi composta da head, 7 standard e tail.
L'immagine head è usata per evitare un'ulteriore differenziazione fra le immagini, già presente fra standard e tail; il suo scopo è quello di rendere l'ultima ALU diversa dalle altre.

2.1 Gestione delle immagini

Per comprendere quali usare, le immagini sono state nominate in base a due differenti codici, uno per le immagini della ALU a 8 bit e l'altro per quelle della ALU a 1 bit.

2.1.0 ALU 8 a bit

La codifica per quanto riguarda i nomi delle immagini della ALU a 8 bit è la seguente:

<lettera_tipo> + <codice_funzione> + “.” + <input> + “.” + <indice_animazione> + “.png”

- <lettera_tipo> è una lettera fra “h”, “s” o “t”, che stanno ad indicare head, standard o tail. Essa serve per distinguere fra i 3 tipi diversi di immagine.

- <codice_funzione> è un stringa di 6 caratteri sull'alfabeto {0,1} e serve ad indicare la funzione specificando i valori delle 6 linee di controllo della ALU.

I codici sono quelli presenti nella seguente tabella:

F_0	F_1	ENA	ENB	INVA	INC	Funzione
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

- <input> è un stringa di 2 caratteri sull'alfabeto {0,1} ed indica l'input A e l'input B. Come è possibile vedere le immagini di tipo head non prevedono questi 2 caratteri e neanche il punto che li precede.
- <indice_animazione> è un carattere che può essere '0' o '1'. '0' indica lo stato della singola ALU prima che calcoli la funzione, '1' indica lo stato della ALU dopo aver calcolato la funzione (e quindi con un eventuale output 1 o un riporto in uscita, ecc...).
  Nel caso in cui una data animazione non preveda differenze fra lo stato iniziale e quello finale, l'immagine con indice '1' non sarà presente.

Ecco alcuni esempi:

- Il codice dell'immagine head relativo alla funzione A+B è “h111100.0.png”.
- Lo stesso relativo all'immagine standard nell'ipotesi di input 1,0 e prima che calcoli la funzione è “s111100.10.0.png”.
- Mentre per l'immagine tail è “t111100.10.0.png”.

ATTENZIONE:
Come detto prima: il codice della funzione nel nome del file rispecchia esattamente lo stato delle linee di controllo delle ALU.
Vi sono però funzioni che prevedono la linea INC della prima ALU asserita, ciò però non vuol dire che necessariamente tutte le altre ALU debbano avere INC asserito.
Per esempio nella funzione A+B+1 la prima ALU avrà sempre codice funzione uguale a “111101”, mentre

è possibile che le altre ALU abbiano codice “111100” anziché il precedente.

2.1.1 ALU a 1 bit

La codifica delle immagini relative ai 5 pezzi della ALU a 1 bit si basa sul seguente schema:

<lettera_parte> + <linea_1> + ... + <linea_n> + “.” + <indice_animazione> + “.png”

- <lettera_parte>: è una lettera che indica a quale delle 5 parti della ALU ci si riferisce:
  - “l”: parte logica;
  - “a”: sommatore;

- “d”: decodificatore;
- “o”: output;
- “r”: parte di “raccordo”;

- <linea_1>, …, <linea_n>: sono le singole linee di input delle 5 parti :

  – parte logica: abbiamo 5 linee che indicano in ordine: INVA, ENA, A, ENB, B.
    Esempio: 01111 indica l'abilitazione e la presenza di entrambi gli input senza l'inversione dell'input A.
  – sommatore: abbiamo 4 linee che indicano in ordine: il riporto in ingresso, l'input B, l'input A, il segnale di abilitazione del decodificatore.
    Esempio: 1110 indica la presenza di riporto in ingresso, degli input A e B e l'assenza del segnale di abilitazione della somma.
  – decodificatore: abbiamo solo 2 linee: una indica F0 e l'altra F1.
  – output: abbiamo 5 linee che indicano in ordine: il riporto in ingresso, gli output delle operazioni di AND, OR, NOT e somma.
    Esempio: 01000 indica che l'output dell'operazione AND è 1.
  – parte di “raccordo”: abbiamo 5 linee che in ordine indicano: l'input A, l'input B, i segnali di abilitazione delle operazioni di AND, OR e NOT.
    Esempio: 11000 indica che sia l'input A che B sono presenti e che scelta è la somma (la cui linea di abilitazione non si trova in questo parte di immagine della ALU).

- <indice_animazione>: indica lo stato di progressione dell'animazione per quella parte di ALU. Si parte dall'indice 0 fino ad arrivare ad un indice che varia per le singole parti.

  ATTENZIONE: l'indice non è del tutto progressivo, dato che certe parti prevedono comportamenti diversi in base alle funzioni.
  Esempio: vi sono animazioni che hanno come primo indice 0 e poi, anziché passare all'indice 1, salto all'indice 5.
  Per accertarsi quale sia l'ultimo indice di una data animazione controllare direttamente le immagini.

3. Organizzazione della grafica

La componente grafica del programma è piuttosto semplice: essa è composta da un JFrame contenente due JLayeredPane, che contengono l'ALU a 1 bit e quella a 8 bit, una JComboBox per la scelta della funzione e 4 JButton per il controllo dell'animazione.

Le varie immagini delle ALU altro non sono che dei JLabel i quali, modificando la proprietà “icon”, consentono di visualizzare in maniera semplice e veloce l'immagine scelta.

La scelta di usare i JLayeredPane anziché i JPanel è stata obbligata, in quanto vi era la necessità che i singoli JLabel, contenenti le immagini, combaciassero in maniera perfetta in modo da sembrare un'unica immagine.

4. Architettura del software

Dato che il compito di questo programma è incentrato nella sola gestione delle immagini nei vari JLabel, si è optato, in quanto ritenuto meno dispersivo e più efficiente, di inglobare tutti i metodi necessari in un'unica classe.
Tale classe è Alu.java.

Mettendo da parte i metodi per la gestione dell'input, il core del programma consiste nei metodi runFunction(), paintSequence(int _offset) e nei vari metodi load (loadAnd, loadOr, ecc.).
Hanno inoltre importanza cruciale al funzionamento dei suddetti metodi le variabili globali: String sequence[][], int sequenceIndex, int aluIndex, String byteA, String byteB.

Il processo di funzionamento è il seguente:

una volta scelto l'input, la funzione da calcolare e premuto il pulsane “Avvia simulazione”, viene chiamata la procedura runFunction(). Tale procedura inizializza sequenceIndex, aluIndex, byteA e byteB, quindi a seconda della funzione scelta richiama uno dei metodi load. Ogni metodo load non fa altro che caricare in sequence[][] (una matrice 5*n, dove n è la lunghezza dell'animazione) i nomi delle immagini che andranno a caratterizzare ogni singolo stato della ALU a 1 bit (partendo dalla prima ALU fino all'ultima).
Una volta caricata l'animazione, ogni qualvolta si premerà il pulsante “Avanti” o “Indietro”, verrà chiamato il metodo paintSequence(int _offset) che oltre a mandare avanti (o indietro) l'animazione gestisce lo stato della ALU a 8 bit. Per fare ciò questo metodo si appoggia alle variabili globali sequenceIndex, che indica l'indice della sequenza correntemente visualizzata e aluIndex, che è l'indice dell'ALU correntemente in uso.
Le variabili byteA e byteB sono le 2 parole da 8 bit sulle quali calcolare la funzione.

5. Possibili miglioramenti

Da un punto di vista delle animazioni non vedo un possibile ulteriore sviluppo, in quanto sono simulate sotto ogni aspetto tutte le funzioni calcolabili da una ALU.
Per quanto riguarda l'interfaccia grafica sarebbe auspicabile un restyling della versione LD. Nel fare ciò raccomando di mantenere la semplicità di uso che contraddistingue questa versione.
Un altro possibile miglioramento potrebbe consistere in una ristrutturazione del codice da un punto di vista della gestione delle animazioni, e quindi anche in una migliore riscrittura del metodo paintSequence(int _offset), magari scindendolo in più metodi per renderlo più chiaro.
Non ritengo utile l'aggiunta della funzione di avanzamento automatico dell'animazione a scatti temporali predeterminati, in quanto (per collaudo effettuato) ciò non giova alla comprensione del funzionamento interno della ALU a 1 bit.

6. Distribuzione del software

VISUAL 8b, insieme alla sua documentazione, è rilasciato sotto licenza GPL e può quindi essere distribuito e modificato senza limitazioni.

Per eventuali dubbi relativi alla modifica del programma che non dovessero trasparire da questa

guida o dai commenti al codice sorgente, o altro, è possibile contattarmi a questo indirizzo federico.vindigni@gmail.com