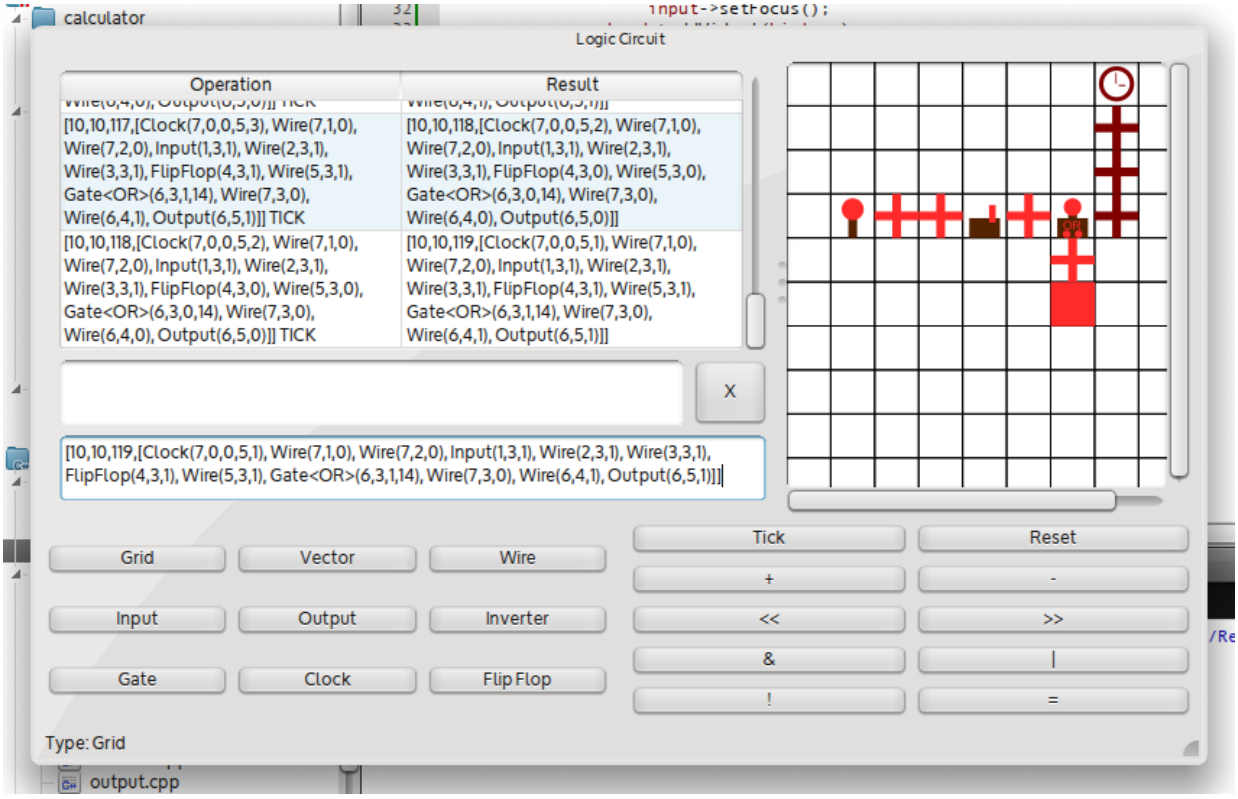
Federico Rampazzo
Matricola: 1001352
Anno accademico: 2011/2012

Relazione progetto

Calcolatrice Estendibile: Circuiti Logici



Introduzione:

Il progetto Circuiti Logici è un programma che consente all'utente di creare, modificare e operare griglie logiche - dove con griglie logiche si intende una collezione ordinata di celle caratterizzate da un segnale binario (0|1) in grado di propagarsi tra celle in maniera unica e dipendente dalle tipologie di celle.

Federico Rampazzo
Matricola: 1001352
Anno accademico: 2011/2012

Informazioni sullo sviluppo:

Il progetto è stato realizzato su Arch Linux i686, gcc 4.6.3 e Qt 4.8.0.

Informazioni sulla compilazione:

Per compilare il progetto è sufficiente generare un Makefile appropriato utilizzando il comando *qmake*, e lanciare *make*.

Oggetti utilizzabili:

L'utente è in grado di creare una griglia eseguendo una serie di operazioni tra gli oggetti disponibili.

Grid(larghezza, altezza, istanti, celle[ ]):

L'oggetto Grid rappresenta una collezione di celle disposte su un piano bidimensionale.
È caratterizzato da larghezza e altezza della griglia e dal numero di istanti (detti anche ticks) che sono passati dalla creazione dell'oggetto.

Input(x, y, segnale):

L'oggetto input (inserito nella griglia in posizione *x*, *y*) genera un *segnale* 1 e lo propaga alle celle circostanti. È una delle poche celle in grado di generare un segnale.

Wire(x, y, segnale):

L'oggetto Wire si limita a propagare il *segnale* che riceve alle celle circostanti.

Inverter(x, y, segnale):

L'oggetto Inverter inverte il *segnale* che riceve e lo trasmette alle celle circostanti.

Output(x, y, segnale):

L'oggetto Output viene utilizzato unicamente per osservare il *segnale* risultante da un circuito (o da una parte di esso).

FlipFlop(x, y, segnale):

L'oggetto FlipFlop si limita a invertire il suo *segnale* ogni volta che riceve un *segnale* 1.
Un array di FlipFlop può essere utilizzato per costruire una basilare memoria (in cui ogni FlipFlop rappresenta un bit).

Gate(x, y, segnale, simbolo):

L'oggetto Gate implementa una funzione booleana che accetta in input due *segnali* A e B, calcola il risultato in base al valore di *simbolo* e propaga il risultato.
Se *simbolo* è & verrà effettuata l'operazione logica AND, se invece è | verrà effettuata l'operazione logica OR.
Partendo da questi due simboli è possibile effettuare operazioni NOT, AND oppure OR tra Gates, così da ottenere un Gate che implementa NAND, NOR, XOR, XNOR.
I Gates che possono essere generati tramite operazioni tra Gate<AND> e Gate<OR> utilizzeranno, al posto di *simbolo*, un intero - la cui rappresentazione binaria può essere vista come una tabella di verità in grado di calcolare il risultato di A e B.

Tabella di verità:

A	B	AND	OR	NAND	NOR	XOR	XNOR
0	0	0	0	0	0	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	1	1	0	1

AND: 8=1000, OR: 14=1110, NAND: 7=0111, NOR: 1=0001, XOR: 6=0110, XNOR: 9=1001

Clock(x, y, segnale, periodo, istante):

L'oggetto Clock è la seconda (e ultima) cella generatrice implementata in questo programma.
È caratterizzata da un *periodo* ed emette un *segnale* 1 ogni *periodo* istanti.
Durante gli altri istanti Clock emette un *segnale* 0.

Vector(x, y):

L'oggetto Vector non è una Cella e non può essere inserito dentro ad una griglia.
Viene utilizzato principalmente per shiftare le celle contenute nelle Grid.

Operazioni definite:

Grid << Cell:

Viene inserita Cell in Grid alla posizione x,y specificata in Cell.
Nel caso la posizione non sia valida verrà lanciato un errore "Out of bounds cell".

Grid >> Cell:

Se presente una cella dello stesso tipo di Cell alla posizione x,y in Grid, questa verrà cancellata.
Nel caso la posizione non sia valida verrà lanciato un errore “Out of bounds cell”;
Nel caso la cella sia vuota verrà lanciato un errore “Empty Cell”;
Nel caso la cella sia di un tipo diverso verrà lanciato l’errore “Not the same cell type”.

Grid +- Vector:

Tutte le celle in Grid verranno shiftate di Vector. Nel caso sia necessario Grid verrà estesa
È possibile riscontrare un errore “Out of bounds cell” nel caso l’operazione comporti un risultato non valido.

Grid + Grid:

Le due Grids verranno fuse insieme. Le celle della seconda griglia possono sovrascrivere le celle della prima. Gli errori che possono essere lanciati sono gli stessi di Grid << Cell.

Grid TICK:

Le celle Input e Clock contenute in Grid generano un segnale appropriato che viene propagato tramite tutte le celle adiacenti. Il contatore istanti viene incrementato.
L’unico errore che si può verificare è “Too many inputs for a gate”, nel caso in cui vengano inviati più di due segnali allo stesso Gate.

Grid RESET:

Tutte le celle vengono resettate ai loro valori di default (che non coincidono per forza con i valori iniziali impostati dall’utente).

Vector +- Vector:

È possibile sommare Vettori e Celle a condizione che il tipo sia uguale tra i due oggetti.
Nel caso delle Celle le informazioni contenute nella seconda cella andranno perse.
Nel caso gli oggetti siano diversi verrà lanciato l’errore “Can’t operate on different objects”.

! Gate:

Effettuanto un’operazione NOT su di un Gate la sua tabella di verità viene invertita.

Gate & Gate:

Il risultato è un Gate con una tabella di verità dove il risultato è 1 solo quando è presente in entrambi i Gates.

Gate | Gate:

Il risultato è un Gate con una tabella di verità dove il risultato è 1 quando è presente in almeno uno dei

Gates.

Clock TICK:

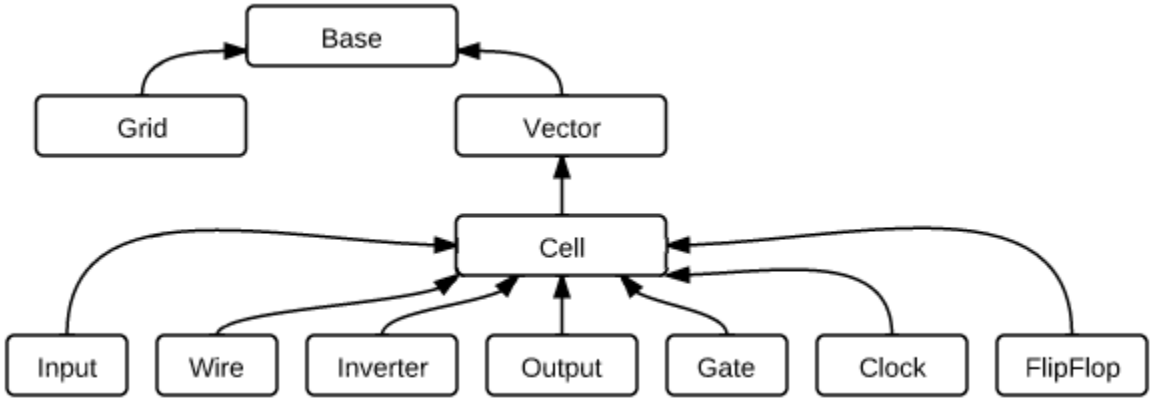
È possibile incrementare un Clock di un istante (in maniera simile a Grid TICK).

Cell RESET:

È possibile resettare il segnale di una cella (in maniera simile a Grid RESET).

I tentativi di utilizzare funzioni non supportate porteranno all'errore "Undefined operation".

Analisi della gerarchia:

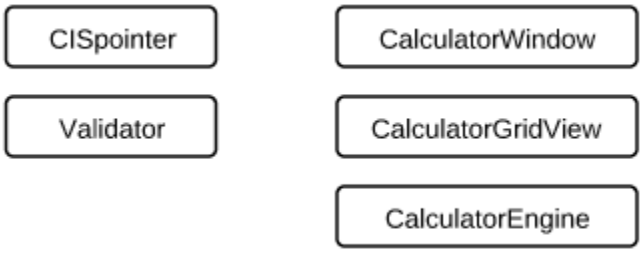


Logica:

Base è una classe astratta, superclasse per tutti gli oggetti che possono essere utilizzati in un'operazione. In Base tutte le funzioni che rappresentano operazioni sono implementate in modo da lanciare un eccezione "Undefined operation".
Ogni sottoclasse reimplementa le operazioni necessarie lasciando non definite le altre; sfruttando il polimorfismo diventa quindi possibile eseguire le operazioni tra tipi diversi in maniera trasparente.
Vector estende Base aggiungendo la gestione delle coordinate (condivisa tra i Vector e le Celle).
Cell è una classe astratta ed estende Vector definendo tramite metodi virtuali la gestione dei segnali necessaria per le celle che faranno parte di Grid
Le classi derivate da Cell implementano i metodi virtuali indicati in Cell e possono quindi soddisfare tutti i requisiti necessari per essere aggiunte in Grid.

Grid contiene un vector (unidimensionale) di CISpointer a Cell che viene utilizzato per salvare in memoria la griglia. Per ogni tick Grid si limiterà a cercare gli Input e i Clock e a generare un segnale.

Questo verrà propagato tramite una funzione ricorsiva che sfrutta alcune funzioni virtuali comuni a tutte le celle.



Utilità:

CISpointer è una classe di Smart Pointers che contano i riferimenti in maniera intrusiva (Counter Intrusive Smart Pointer), sfruttando il parametro references in Base. È implementato come un template <class T> e può essere usato con tutte le classi derivate da Base.

Per risolvere il problema dell'interpretazione dell'input utente ogni classe istanziabile è stata dotata di un costruttore che accetta una stringa e che effettua il parsing costruendo l'oggetto corrispondente - assumendo che la stringa ricevuta sia corretta.
Per verificare l'input è stata invece scritta la classe Validator: questa classe accetta una stringa e un pattern ed estrae dalla stringa i primi caratteri che corrispondono al pattern.
Questa classe tiene conto dell'esigenza di effettuare il parsing di stringhe complesse (che possono contenere a loro volta altri oggetti) e viene usata da CalculatorEngine - il motore che gestisce l'input utente e che genera gli oggetti Base necessari - e Grid (per effettuare il parsing delle celle contenute nel costruttore che accetta string).
Nel caso di Grid diventa quindi possibile provare ad estrarre ogni tipo di cella dalla stringa che contiene le celle contenute in Grid - fino a ottenere una stringa vuota. Se non sarà possibile estrarre tutti i caratteri (salvo le virgole) dalla stringa possiamo considerare non valido l'oggetto.

Calcolatrice:

L'interfaccia utente è stata realizzata creando una sottoclasse di QMainWindow, CalculatorWindow, che si occupa di disegnare i widgets, gestire gli eventi e inviare i comandi dell'utente alla classe CalculatorEngine.

CalculatorEngine riceve le stringhe inserite dall'utente e, dopo averle validate tramite Validator, istanzia gli oggetti corretti, salvando i loro puntatori dentro un vector<CISpointer<Base> >.
Dopo aver ricevuto un operatore e un corretto numero di operandi (per esempio TICK ha bisogno di un operando, << ha bisogno di due operandi) verrà eseguita la corretta operazione tra oggetti Base e l'oggetto risultante verrà restituito a CalculatorWindow.

La scelta di restituire l'oggetto stesso e non soltanto il risultato in una stringa è dettato dalla volontà di disegnare dentro una QGraphicsView alcuni risultati di queste operazioni.
CalculatorWindow infatti non si limita a mostrare il risultato (e salvarlo in un QTableWidget che raccoglie i comandi in ordine cronologico); l'oggetto viene passato ad un widget particolare, un'istanza di CalculatorGridView.

CalculatorGridView è una sottoclasse di QGraphicsView che implementa il rendering dell'oggetto Grid. Tra le funzionalità aggiunte spiccano lo zoom tramite mouse wheel e la gestione del click (utilizzato per selezionare una cella, prima di inserire un nuovo oggetto) in parallelo alla gestione del panning via trascinamento del mouse.
In questo modo l'utente è in grado di vedere chiaramente il risultato delle proprie azioni e può inserire nuovi oggetti con maggiore facilità.

Esempi:

Seguono alcuni esempi di circuiti che dimostrano le funzionalità del programma:

Dimostrazione dei Gates:
[7,6,0,[Clock(0,0,0,2,2), Clock(2,0,0,3,3), Clock(4,0,0,2,2), Clock(6,0,0,3,3), Wire(0,1,0), Gate<AND>(1,1,0,8), Wire(2,1,0), Gate<OR>(3,1,0,14), Wire(4,1,0), Gate<NAND>(5,1,0,7), Wire(6,1,0), Output(1,2,0), Output(3,2,0), Output(5,2,0), Clock(0,3,0,2,2), Clock(2,3,0,3,3), Clock(4,3,0,2,2), Clock(6,3,0,3,3), Wire(0,4,0), Gate<NOR>(1,4,0,1), Wire(2,4,0), Gate<XOR>(3,4,0,6), Wire(4,4,0), Gate<XNOR>(5,4,0,9), Wire(6,4,0), Output(1,5,0), Output(3,5,0), Output(5,5,0)]]

Simulazione di un Gate NOR usando solo Gate NAND:
[10,10,0,[Input(0,0,1), Clock(4,0,0,3,3), FlipFlop(0,1,0), Inverter(4,1,0), Wire(6,1,0), Wire(7,1,0), Wire(0,2,0), Wire(1,2,0), Gate<NOR>(2,2,0,1), Wire(3,2,0), Wire(4,2,0), Wire(5,2,0), Wire(6,2,0), Gate<NAND>(7,2,0,7), Wire(0,3,0), Output(2,3,0), Wire(7,3,0), Wire(0,4,0), Wire(5,4,0), Wire(6,4,0), Wire(7,4,0), Wire(0,5,0), Gate<NAND>(1,5,0,7), Wire(2,5,0), Wire(3,5,0), Wire(4,5,0), Gate<NAND>(5,5,0,7), Wire(0,6,0), Wire(1,6,0), Wire(5,6,0), Gate<NAND>(6,6,0,7), Output(7,6,0), Wire(5,7,0), Wire(6,7,0)]]