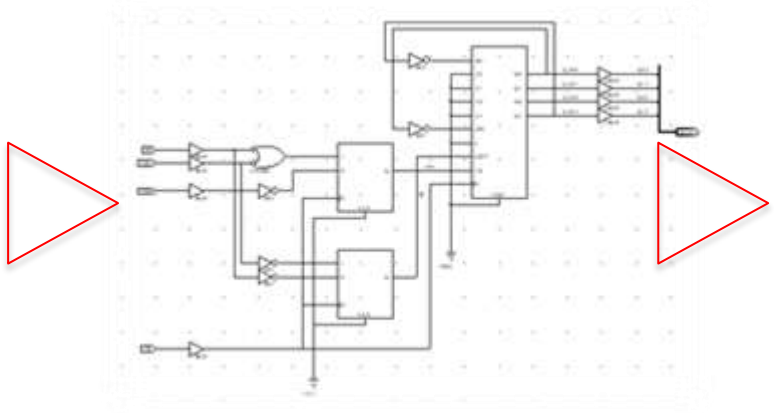
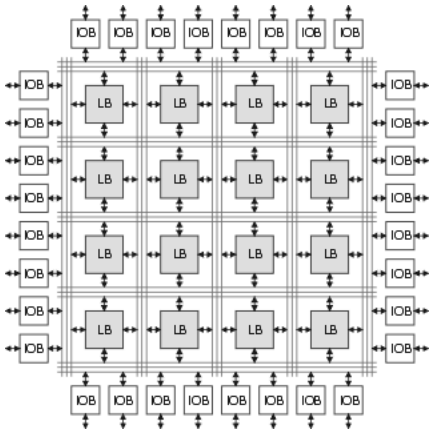
ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA
DEIS - DEPARTMENT OF ELECTRONICS, COMPUTER ENGINEERING AND SYSTEMS

```
begin
  if (RESET_N = '0') then

    for col in 0 to BOARD_COLUMNS-1 loop
    for row in 0 to BOARD_ROWS-1 loop
    ...
   elsif (rising_edge(CLOCK)) then
    ...
```





Laboratorio di Sistemi Digitali M
A.A. 2010/11

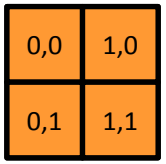
4 – Esercitazione Tetris: Datapath

Primiano Tucci
primiano.tucci@unibo.it
www.primianotucci.com

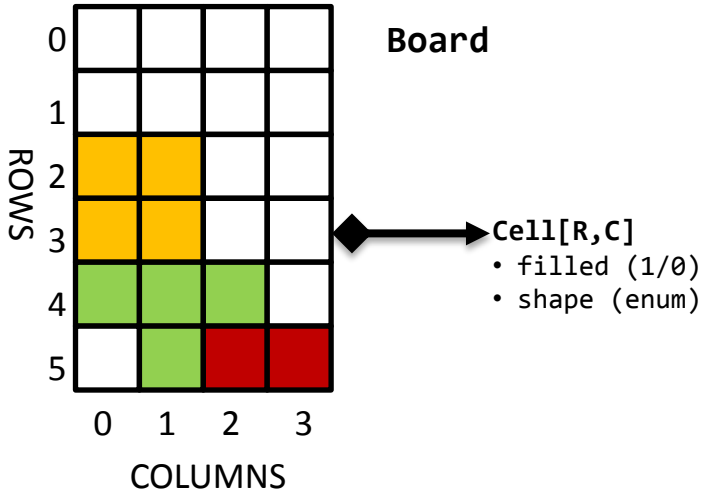
Agenda

1. Identificazione elementi di memoria (registri)
2. Identificazione operazioni sui registri
3. Definizione black-box datapath e segnali scambiati con Control Unit e View
4. Definizione RTL

Precedentemente abbiamo modellato il problema definendo i seguenti tipi di dato:



Piece
- shape (enum)
- blocks[0..3]
  - col (integer 0 to COLS-1)
  - row (integer 0 to ROWS-1)



Quanti registri per i pezzi?

- Registri *board*: 1 (ovviamente!)
- Registri *piece:* **1**

In realtà ci basta mantenere l'informazione solo sul pezzo in caduta: il *falling_piece*

VHDL Tetris_Datapath.vhd (Inizio)

```
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
use work.tetris_package.all;

entity Tetris_Datapath is
    port
    (
          CLOCK            : in  std_logic;
          RESET_N          : in  std_logic;
          -- Connections for the Controller
          ... Le decideremo tra un po' quando le cose saranno più chiare
          -- Connections for the View
          ... Idem
    );
end entity;

architecture RTL of Tetris_Datapath is
    signal board                    : board_type;
    signal falling_piece            : piece_type;
    … dopo ci serviranno altri segnali
begin
    … qui definiremo process e statement concorrenti
```

Il package contenente le costanti e i tipi di dato definiti la settimana scorsa

Per adesso questi segnali non sono ancora dei registri. **Cosa manca?** Il loro RTL! Devono essere assegnati sotto rising_edge!

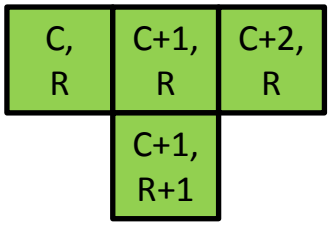
Operazioni sul *falling_piece*

Registro: *falling_piece* ***Tipo di dato:*** *piece_type*
Ruolo: mantiene le informazioni sul pezzo attualmente "in caduta"

Nome operazione	Descrizione	Parametri
NEW_PIECE	Introduce (dall'alto) un nuovo *falling_piece* sostituendo quello precedente	NEW_PIECE_TYPE
MOVE_LEFT	Sposta il *falling_piece* a sinistra	Nessuno
MOVE_RIGHT	Sposta il *falling_piece* a destra	Nessuno
MOVE_DOWN	Sposta il *falling_piece* in basso	Nessuno
ROTATE *	Ruota il *falling_piece* (in senso orario)	Nessuno
CAN_MOVE_LEFT	Determina se il *falling_piece* può essere spostato a sinistra	Nessuno
CAN_MOVE_RIGHT	Determina se il *falling_piece* può essere spostato a destra	Nessuno
CAN_MOVE_DOWN	Determina se il *falling_piece* può essere spostato in basso	Nessuno
CAN_ROTATE	Determina se il *falling_piece* può essere ruotato	Nessuno

* Per semplicità consideriamo solo la rotazione in senso orario

RTL del *falling_piece*



Contenuto del falling_piece. *In caso di:*

•RESET (async) : posizione arbitraria

•**NEW_PIECE** : posizione indicata da NEW_PIECE_TYPE

•Altrimenti : posizione calcolata in funzione del movimento richiesto

Porte (I/O) introdotte

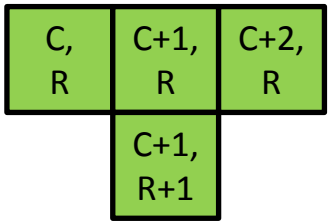
•NEW_PIECE (Input)
•NEW_PIECE_TYPE[…] (Input)

```
FallingPiece_RTL : process(CLOCK, RESET_N)
  constant PIECE_AT_RESET   : piece_type := PIECE_SQUARE;
  constant NEW_PIECE_OFFSET : integer    := BOARD_COLUMNS/2 - 2;
begin

 if (RESET_N = '0') then
    falling_piece <= PIECE_AT_RESET;

 elsif (rising_edge(CLOCK)) then

   if (NEW_PIECE = '1') then
      falling_piece.shape <= NEW_PIECE_TYPE.shape;
      for i in 0 to BLOCKS_PER_PIECE-1 loop
        falling_piece.blocks(i).row <= NEW_PIECE_TYPE.blocks(i).row;
        falling_piece.blocks(i).col <= NEW_PIECE_TYPE.blocks(i).col
                                                    + NEW_PIECE_OFFSET;
      end loop;
    else
      falling_piece <= next_falling_piece;
    end if;

  end if;
end process;
```

Adoperiamo un template sincrono

Decomponiamo il problema.

Generazione combinatoria dei movimenti del *falling_piece*



Porte (I/O) introdotte

- MOVE_DOWN (Input)
- MOVE_LEFT (Input)
- MOVE_RIGHT (Input)
- ROTATE (Input)

```
NextFallingPiece : process(falling_piece, …PORTE_CHE_DEFINIREMO…)
begin
  next_falling_piece <= falling_piece;

  for i in 0 to BLOCKS_PER_PIECE-1 loop

    if (MOVE_DOWN = '1') then
      next_falling_piece.blocks(i).row <= falling_piece.blocks(i).row + 1;
    elsif (ROTATE = '1') then
          SCOPRITELO! (rispolverate le conoscenze di geometria)
    elsif (MOVE_LEFT = '1') then
      next_falling_piece.blocks(i).col <= falling_piece.blocks(i).col - 1;
    elsif (MOVE_RIGHT = '1') then
      next_falling_piece.blocks(i).col <= falling_piece.blocks(i).col + 1;
    end if;

  end loop;
end process;
```

Confermo la posizione corrente, salvo smentita (più sotto).

Ragioniamo per ogni singolo blocchetto

"Smentisco" la posizione verticale

"Smentisco" la posizione orizzontale

Che succede se invece degli elsif ci sono degli end if ?

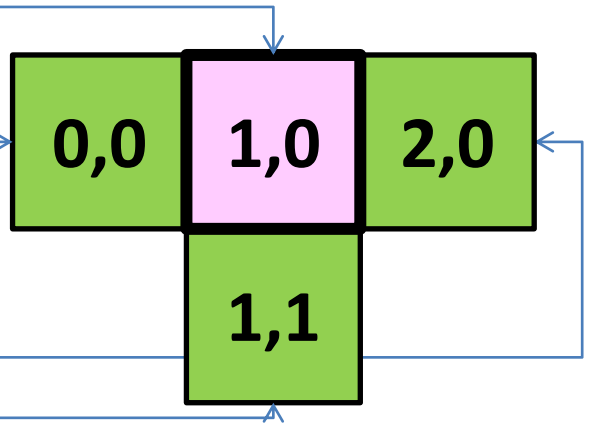
… e se inverto il for e gli if ?

I segnali MOVE_* (&c.) provengono dal controller NON dai pulsanti.

Rotazione del falling_piece

- Ricordate come abbiamo definito i pezzi nel package?

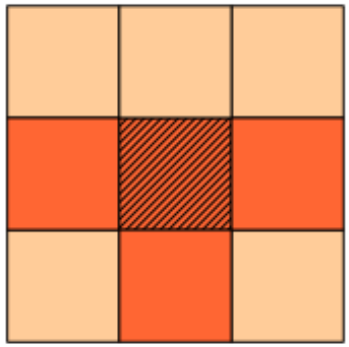
```
constant PIECE_T : piece_type := (
  shape  => SHAPE_T,
  blocks => (
      (col => 1, row => 0),
      (col => 0, row => 0),
      (col => 2, row => 0),
      (col => 1, row => 1)
      ));
```



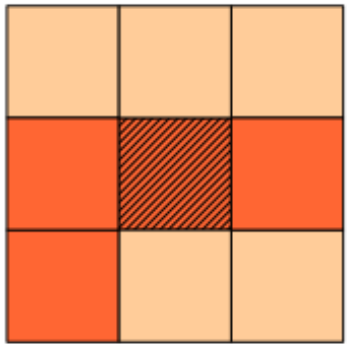
- Ma, in fondo, l'ordine in cui vengono definiti i 4 blocchi è irrilevante.
- Facciamo una piccola variazione: il primo dei 4 blocchi rappresenta il centro di rotazione del pezzo: il pivot
- Il pivot, per definizione, è il punto fisso di una rotazione.

Pivot dei pezzi (courtesy of Marco Allegretti)

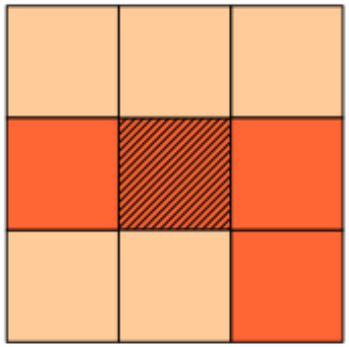
SHAPE_T



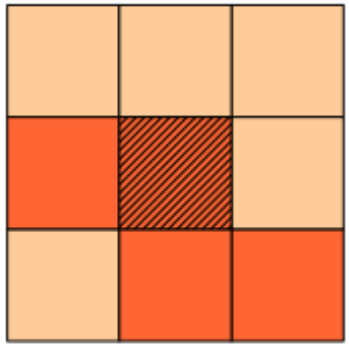
SHAPE_L_L



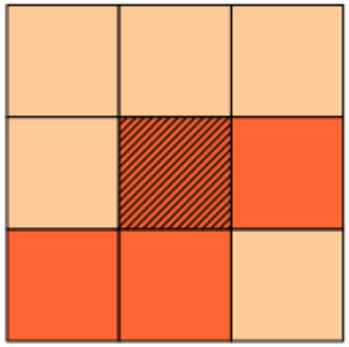
SHAPE_L_R



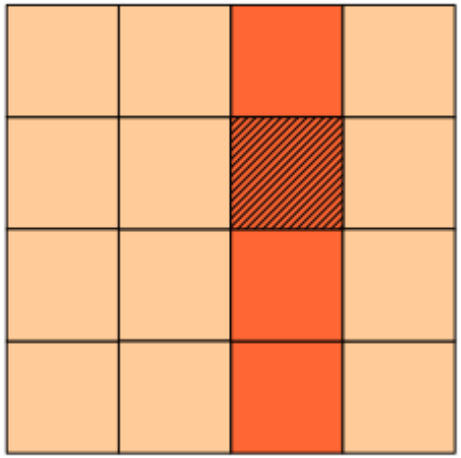
SHAPE_DOG_R



SHAPE_DOG_L



STICK



Rotazione del falling_piece

- Il pivot non cambia le sue coordinate durante una rotazione.
- Ed i rimanenti 3 blocchi? Intuitivamente, nel caso di rotazione oraria:
  - Tanto più un blocco è più in basso del pivot, tanto più si troverà alla sua sinistra.
  - Tanto più un blocco è a sinistra del pivot, tanto più si ritroverà in alto.
  - Parimenti per alto e sinistra.

Per ogni blocco i-esimo:

$x_{offset\_i} = (x_i - x_p)$

$y_{offset\_i} = (y_i - y_p)$

(possono assumere valori negativi!)

$x_{new} = x_p - y_{offset\_i} = x_p - y_i + y_p$

$y_{new} = y_p + X_{offset\_i} = y_p + x_i - x$

VHDL per rotazione falling_piece

```
NextFallingPiece : process(falling_piece,MOVE_DOWN,MOVE_LEFT,MOVE_RIGHT,ROTATE)
  variable pivot : block_pos_type;
begin
  next_falling_piece <= falling_piece;
  pivot              := falling_piece.blocks(0);

  for i in 0 to BLOCKS_PER_PIECE-1 loop
  ...
    elsif (ROTATE = '1') then
      if (i /= 0) then -- the pivot does not require any transformation
        next_falling_piece.blocks(i).col <=
          pivot.col - (falling_piece.blocks(i).row - pivot.row);

        next_falling_piece.blocks(i).row <=
          pivot.row + (falling_piece.blocks(i).col - pivot.col);
      end if;
   ...
```

Generazione combinatoria dei segnali CAN_MOVE_X

0	C, R	C+1, R	C+2, R
0	1	C+1, R+1	0
0	0	1	0

Porte (I/O) introdotte

- CAN_MOVE_DOWN (Output)
- CAN_MOVE_LEFT (Output)
- CAN_MOVE_RIGHT (Output)
- CAN_ROTATE (Output)

```
CanMove_Signals : process(falling_piece, board)
  variable cur_block          : block_pos_type;
  variable left_cell_filled   : std_logic;
  ...
begin
  CAN_MOVE_LEFT  <= '1';
  CAN_MOVE_RIGHT <= '1';
  CAN_MOVE_DOWN  <= '1';
  CAN_ROTATE     <= '1';

  for i in 0 to BLOCKS_PER_PIECE-1 loop
    cur_block := falling_piece.blocks(i);
    if (cur_block.col = 0) then
      CAN_MOVE_LEFT <= '0';
    else
      left_cell_filled := board.cells((cur_block.col-1),cur_block.row).filled;
      if (left_cell_filled = '1') then
        CAN_MOVE_LEFT <= '0';
      end if;
    end if;
    . . .
    RIGHT, e DOWN omissis (sono praticamente identici)
    ROTATE !
end process;
```

Il movimento del *falling_piece* è libero in tutte direzioni… salvo smentita (più sotto)

Ragioniamo sempre per ogni singolo blocco

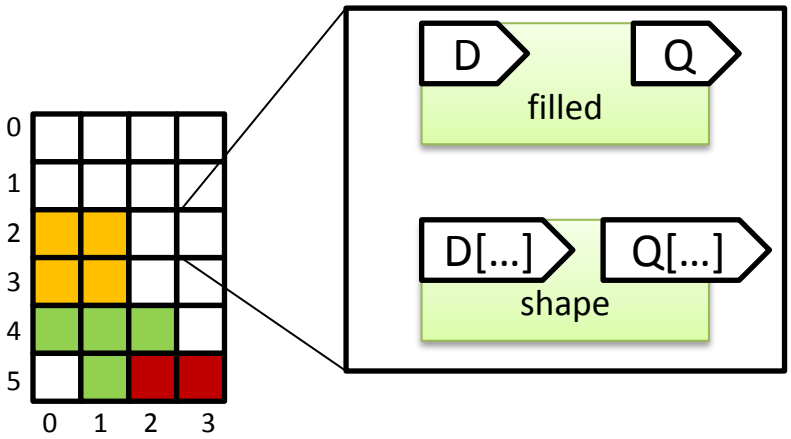
Se uno qualsiasi dei (4) blocchi che compongono il pezzo è in colonna 0, il pezzo non può andare più a sinistra di così. Ma non basta…

Se uno (o più) dei (4) blocchi è ostruito a sinistra, il pezzo non può muoversi a sinistra.

Datapath :: board

Registro *board* ***Tipo di dato:*** *board_type* Ruolo: mantiene le informazioni sull'evoluzione del gioco (pezzi caduti)		
Nome operazione	**Descrizione**	**Parametri**
CLEAR	Svuota la board liberando tutte le celle	nessuno
MERGE	Fonde il *falling_piece* nella board occupando le celle relative alla sua posizione attuale	Nessuno
REMOVE_ROW	"Elimina" la riga selezionata. Di fatto copia il contenuto della riga soprastante.	ROW_INDEX (integer)
CHECK_ROW	Verifica il completamento della riga selezionata	ROW_INDEX (integer)
QUERY_CELL	Recupera il contenuto di una cella (per consentirne il disegno… vedremo dopo)	POSIZIONE (block_pos_type)

RTL della *board (1/3)*



Contenuto delle celle della *board. In caso di:*

- •RESET (async) : filled = 0
- •CLEAR : filled = 0
- •REMOVE_ROW : contenuto cella soprastante (!)
- •MERGE : eventuale blocco del falling_piece avente coordinate corrispondenti
- •Alrimenti : mantiene il valore corrente

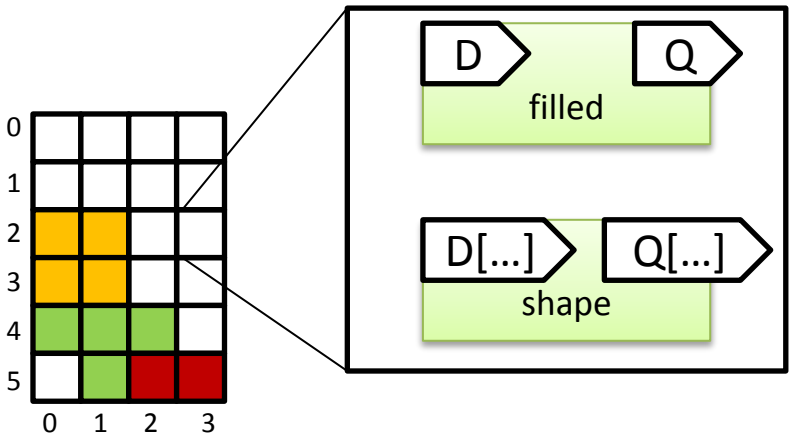
Porte (I/O) introdotte

- •REMOVE_ROW (Input)
- •ROW_INDEX[…] (Input)

```
Board_RTL : process(CLOCK, RESET_N)
begin
  if (RESET_N = '0') then

    for col in 0 to BOARD_COLUMNS-1 loop
      for row in 0 to BOARD_ROWS-1 loop
        board.cells(col,row).filled <= '0';
      end loop;
    end loop;

  elsif (rising_edge(CLOCK)) then

   for col in 0 to BOARD_COLUMNS-1 loop
    for row in 0 to BOARD_ROWS-1 loop
      RAGIONIAMO CELLA PER CELLA!
     if (CLEAR = '1') then
       ... (continua nella prossima slide)
     elsif (REMOVE_ROW = '1') then
       ... (continua nella prossima slide)
     elsif (MERGE = '1') then
       ... (continua nella prossima slide)
     end if;
    end loop; -- row loop
   end loop;  -- col loop
  end if;  -- if rising_edge(..)
end process;
```

Nota: CLOCK e RESET_N impliciti per tutti i registri. RESET_N (asincrono) va utilizzato solo ed esclusivamente per l'inizializzazione dell'hardware e non per espletare operazioni (sincrone)

RTL della *board (2/3)*



Contenuto delle celle della *board. In caso di:*

- •RESET (async) : filled = 0
- •**CLEAR** : filled = 0
- •**REMOVE_ROW** : contenuto cella soprastante (!)
- •**MERGE** : eventuale blocco del falling_piece avente coordinate corrispondenti
- •Alrimenti : mantiene il valore corrente

Aggiungiamo questi segnali alla entity (prima di begin)

```
type affected_by_merge_type is
  array(natural range <>, natural range <>) of std_logic;

Signal cell_ffected_by_merge : affected_by_merge_type
    (0 to BOARD_COLUMNS-1, 0 to BOARD_ROWS-1);
```

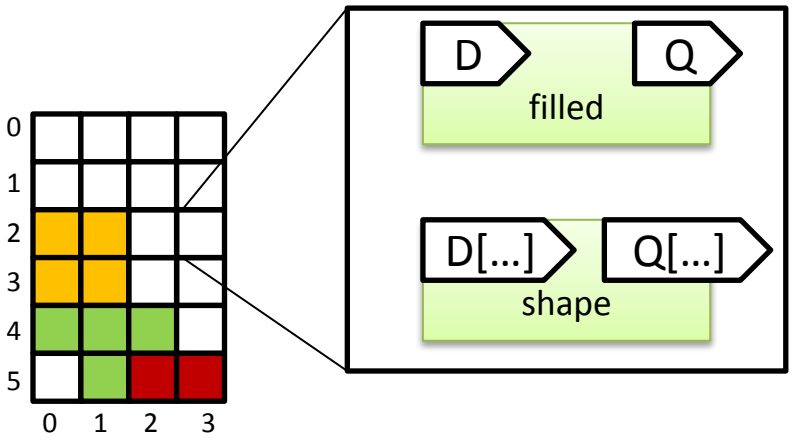
LOGICA RTL *(continua da slide precedente)*

```
for col in 0 to BOARD_COLUMNS-1 loop
for row in BOARD_ROWS-1 downto 0 loop

  if (CLEAR = '1') then

      board.cells(col,row).filled <= '0';

  elsif (REMOVE_ROW = '1') then

   if (row = 0) then
     board.cells(col, row).filled <= '0';
   elsif (row <= ROW_INDEX) then
     board.cells(col, row) <= board.cells(col, row-1);
   end if;

  elsif (MERGE = '1') then

   if(affected_by_merge(col, row) = '1') then
    board.cells(col, row).filled <= '1';
    board.cells(col, row).shape <=falling_piece.shape;
   end if;

  end if;
end loop; -- row loop
end loop; -- col loop
```

Divide et Impera

Quando il problema non è di immediata soluzione, delegatelo … a voi stessi!

RTL della *board (3/3)*



Contenuto delle celle della *board. In caso di:*

- RESET (async) : filled = 0
- **CLEAR** : filled = 0
- **REMOVE_ROW** : contenuto cella soprastante (!)
- **MERGE** : eventuale blocco del falling_piece avente coordinate corrispondenti
- Altrimenti : mantiene il valore corrente

Generazione segnale (array) affected_by_merge

Ruolo: *indica, per ogni cella, se essa è interessata dall'operazione merge, ovvero se uno dei blocchi elementari del falling_piece ne occupa la posizione.*

```
type affected_by_merge_type is
  array(natural range <>, natural range <>) of std_logic;

signal cell_affected_by_merge : affected_by_merge_type
                (0 to BOARD_COLUMNS-1, 0 to BOARD_ROWS-1);

-----------------------
begin (riferito ad: architecture RTL of Tetris_Datapath)
---------------------

AffectedByMerge : process(board, falling_piece)
begin
  cell_affected_by_merge <= ((others=> (others=>'0')));

  for i in 0 to BLOCKS_PER_PIECE-1 loop
    cell_affected_by_merge(
      falling_piece.blocks(i).col,
      falling_piece.blocks(i).row
    ) <= '1';
  end loop;

end process;
```

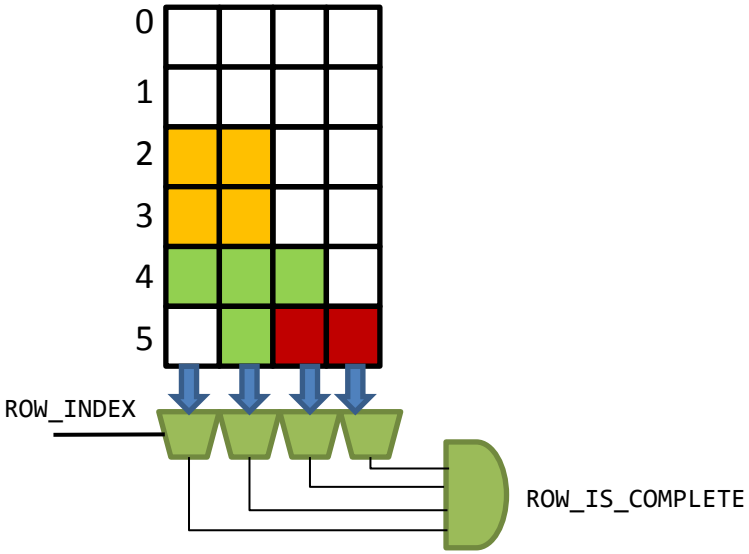
Usiamo un *process* combinatorio per descrivere la logica più complessa (in fondo mica tanto) del segnale (in realtà un array) *affected_by_merge*

Generazione combinatoria di ROW_IS_COMPLETE

```
RowCheck : process(board_r, ROW_INDEX)
  variable complete : std_logic;
begin
  complete := '1';
  for i in 0 to (BOARD_COLUMNS-1) loop
    complete := complete and board.cells(i,ROW_INDEX).filled;
  end loop;
  ROW_IS_COMPLETE <= complete;
end process;
```

Ad un certo punto il controller dovrà verificare se abbiamo completato una riga!

In questo caso la variabile viene usata per generare dinamicamente (in funzione di BOARD_COLUMNS) una catena di AND.



Porte (I/O) introdotte

- ROW_INDEX (Input)
- ROW_IS_COMPLETE (Output)

Operazione di QUERY_CELL (combinatoria)

Utilizzata dal View per disegnare le varie celle

Attenzione!

Nella nostra modellazione la board NON contiene il falling_piece (lo conterrà in occasione del successivo MERGE).

Ma

Il View non lo sa (e non lo vuole sapere).
Il View ha il compito di disegnare la scena. Per farlo non fa altro che interrogarci, chiedendoci lo stato di una determinata cella.
Spetta a noi fornirgli il quadro complessivo (board + falling piece)

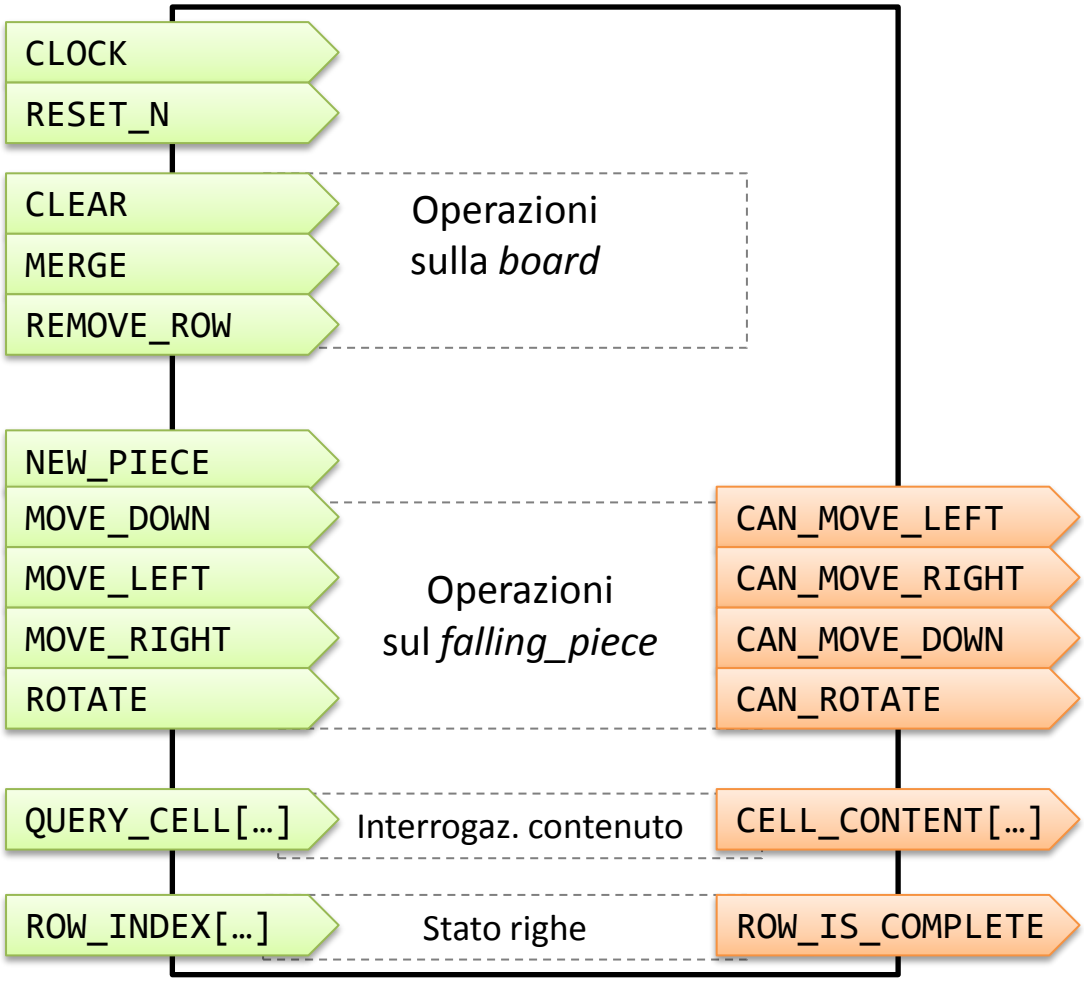
Porte (I/O) introdotte

- QUERY_CELL (Input), di tipo block_pos_type {col, row}
- CELL_CONTENT (Output)

```
CellQuery : process(QUERY_CELL, board, falling_piece)
  variable selected_cell : board_cell_type;
begin
  CELL_CONTENT.filled <= '0';
  CELL_CONTENT.shape  <= SHAPE_T;
  -- indifferente, ma nei processi combinatori
  -- esplicitate sempre tutti gli assegnamenti.

  selected_cell := board.cells(QUERY_CELL.col, QUERY_CELL.row);

  CELL_CONTENT <= selected_cell;
  -- Innanzitutto prendo la cella selezionata dalla board.
  -- In più faccio override in caso uno dei blocchi del
  -- falling_piece occupi la posizione selezionata.
  for i in 0 to BLOCKS_PER_PIECE-1 loop
   if(falling_piece.blocks(i) = QUERY_CELL) then
     CELL_CONTENT.filled <= '1';
     CELL_CONTENT.shape  <= falling_piece.shap
    end if;
  end loop;

  end if;
end process;
```

Come mai non c'è un sengale QUERY (std_logic) ma solo l'indirizzo della cella da interrogare?

Datapath - Out of the box



Datapath - Out of the box (VHDL)

```
entity Tetris_Datapath is
  port
  (
    CLOCK           : in  std_logic;
    RESET_N         : in  std_logic;

    -- Connections for the Controller
    CLEAR           : in  std_logic;
    MOVE_DOWN       : in  std_logic;
    MOVE_LEFT       : in  std_logic;
    MOVE_RIGHT      : in  std_logic;
    ROTATE          : in  std_logic;
    MERGE           : in  std_logic;
    REMOVE_ROW      : in  std_logic;
    NEW_PIECE       : in  std_logic;
    NEW_PIECE_TYPE  : in  piece_type;
    ROW_INDEX       : in  integer
                      range 0 to (BOARD_ROWS-1);
    CAN_MOVE_LEFT   : out std_logic;
    CAN_MOVE_RIGHT  : out std_logic;
    CAN_MOVE_DOWN   : out std_logic;
    CAN_ROTATE      : out std_logic;
    ROW_IS_COMPLETE : out std_logic;
    -- Connections for the View
    QUERY_CELL      : in
       block_pos_type;
    CELL_CONTENT    : out
       board_cell_type
    );

end entity;
```