Implementing Logic

Register Transfer Level

Implementing Logic

- Previously we examined the basic building blocks of digital circuits.
  - Adders and Multipliers
  - Decoders and Multiplexers
  - Latches and Flip-Flops
  - Registers, Counters and Synchronous circuits
  - Memory Circuits

- How are large circuits incorporating all of these created?

Implementing Logic

- The state table for a large circuit containing multiple sub-circuits is contains a large number of states and becomes unmanageable.
- To avoid this difficulty, large circuits are designed in a **modular** way.
- A large design is partitioned into modules.
- Modular subsystems may be arithmetic units, registers, multiplexers, etc.
- The modules are connected with data and control path to form the digital system.

Register Transfer Level

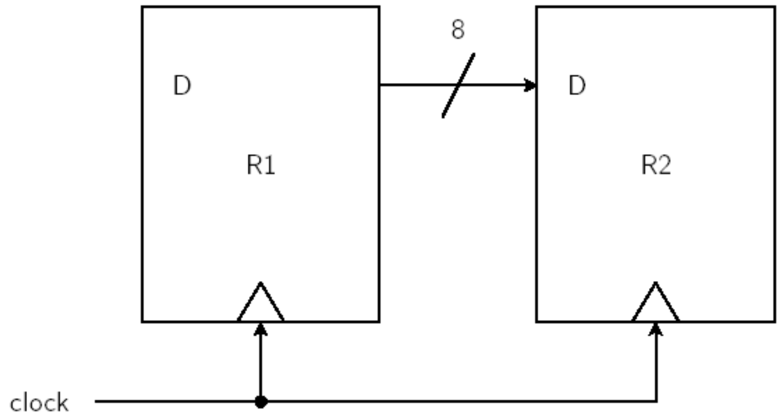
- The digital modules are typically described in terms of registers.
- Each module is described in terms of the registers used and the operations performed.
- The data flow and processing operations are referred to as register transfer operations.
- A **register transfer level** description of a digital system is described in terms of
  - the set of registers in the system
  - the operations performed by the registers
  - the control that supervises the sequence of operations

Register Transfer Level

- Information transfer from one register to another is designated by the operation

$$R2 \leftarrow R1$$

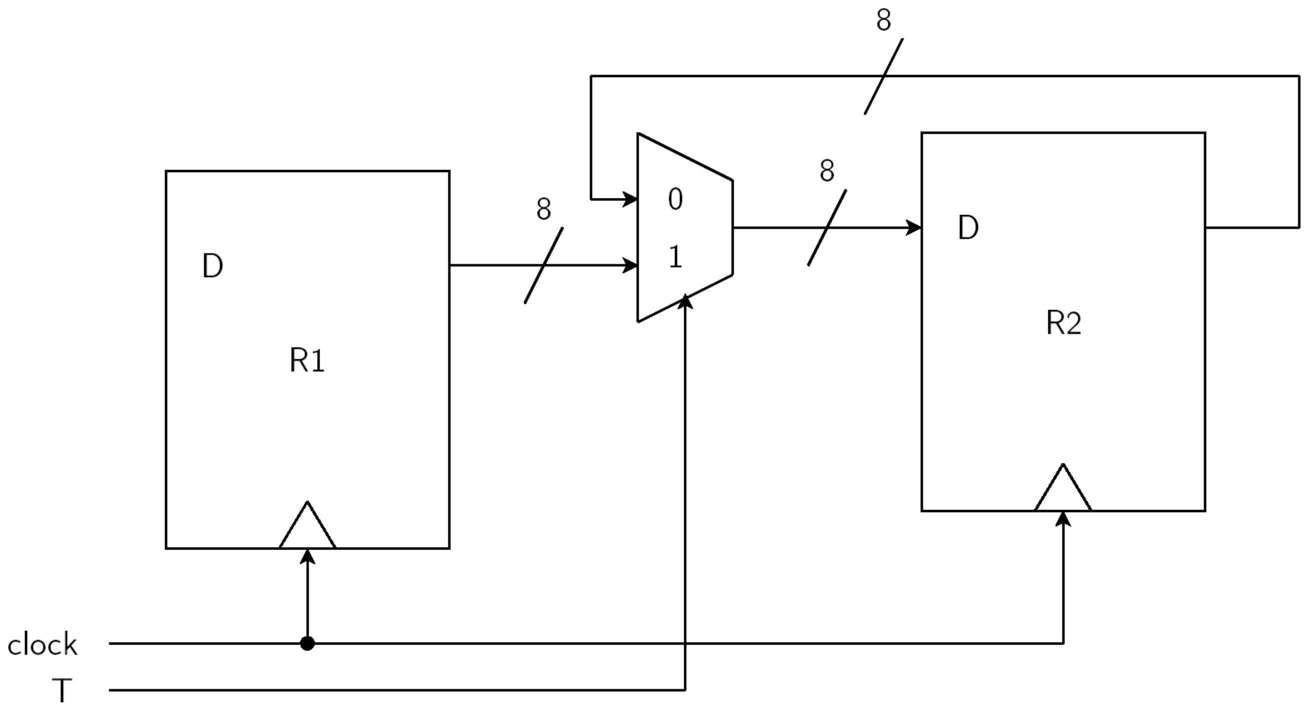
- Typically we want operations to occur on a synchronised clock.



Register Transfer Level

- Conditional statements are also used

$$\textbf{if}\ (\mathsf{T} = 1)\ \textbf{then}\ (\mathsf{R2} \leftarrow \mathsf{R1})$$

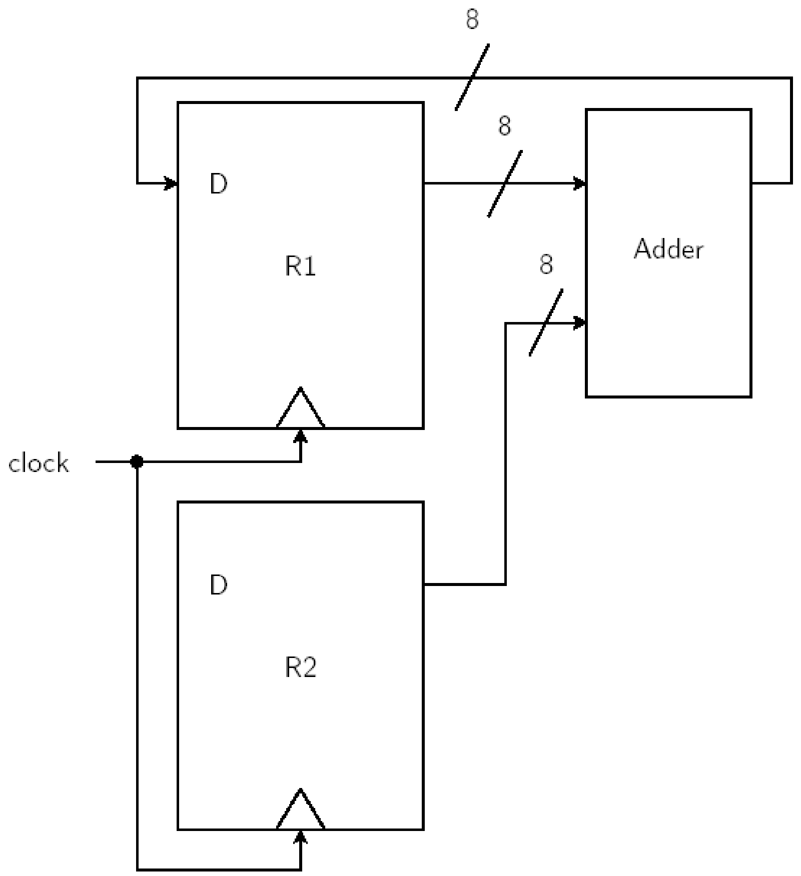


Register Transfer Level

- Addition circuits are easily represented

$$R1 \leftarrow R1 + R2$$

- The output of the adder must be truncated.



Register Transfer Level

- The various register transfer operations can be easily represented using a symbolic language.

$\mathsf{R3} \leftarrow \mathsf{R3} + 1$	increment R3 by 1
$\mathsf{R4} \leftarrow \mathsf{rshift(R4)}$	shift R4 to the right
$\mathsf{R5} \leftarrow 0$	clear R5
$\mathsf{R6} \leftarrow \mathsf{R6\ AND\ 0100}$	R6 is ANDed with 0100

Register Transfer Level

- Can a symbolic language be used to completely describe the operation of digital circuits?
- Such a language would need to describe the register transfer level operations

1. Transferring data from one register to another
2. Arithmetic operations on data in registers
3. Logic operators
4. Shift operators

Hardware Description Languages

- Register Transfer Level operations can be represented symbolically using a **hardware description language** (HDL).
- Two main HDL standards are used today
  - Verilog HDL
  - VHDL (Very High Speed Integrated Circuit HDL)
- Used for
  - circuit modelling
  - simulation
  - and synthesis.

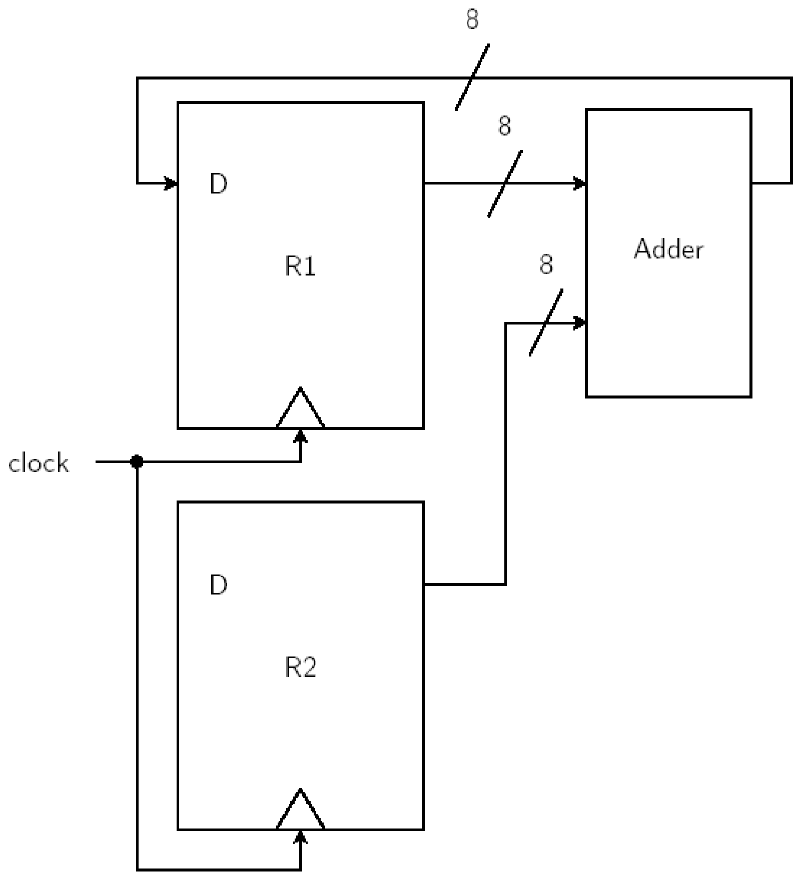
Hardware Description Languages

- Verilog description of

$$R1 \leftarrow R1 + R2$$

is given as

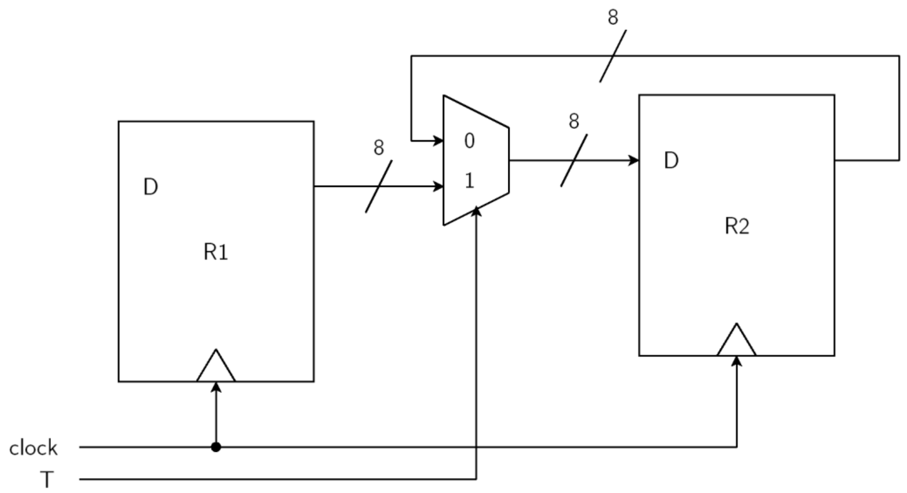
```
always @ (posedge clock)
  begin
    R1 <= R1 + R2;
  end
```



Hardware Description Languages

- Multiplexers are represented using Verilog conditional statements

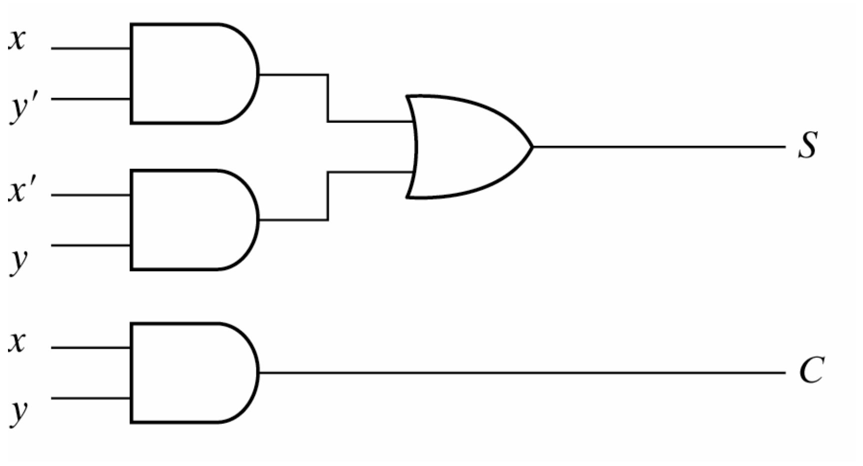
```
always @ (posedge clock)
   if (T == 1)
      R2 <= R1;
   else
      R2 <= R2;
```



Hardware Description Languages

- Combinational logic in Verilog

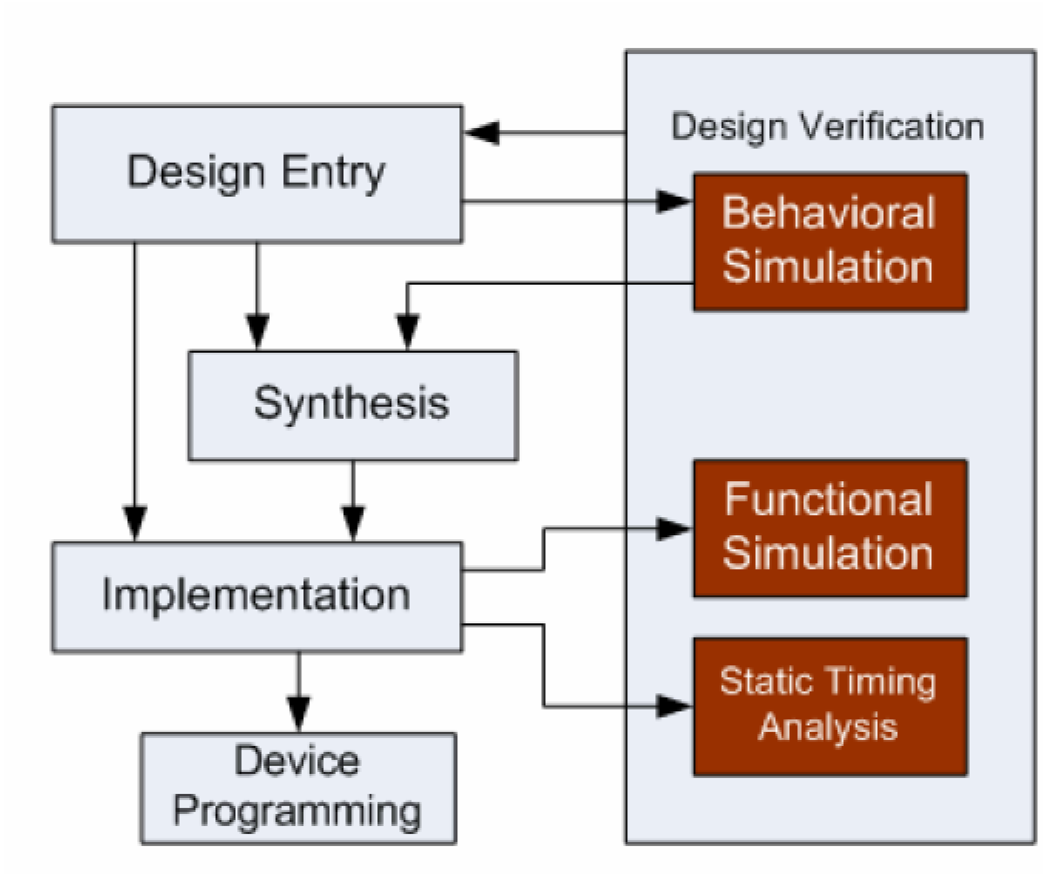
```
always @ (x or y)
  begin
    S <= A | B;
    A <= x & ~y;
    B <= ~x & y;
    C <= x & y;
  end
```



Hardware Description Languages

- We have examined some basic Verilog descriptions.
- HDLs were used originally to model and test digital designs, and then used to implement logic…
- All modern ASICs (Application Specific ICs) and FPGAs (Field Programmable Gate Arrays) implementations are created using a HDL.
- You will learn more about them in the final-year EE module “Digital System Design”.

FPGA Design Flow



FPGA Design Flow

- Design Entry
  - Schematic or HDL languages
- Synthesis
  - Translate HDL code to an optimized netlist of gates that perform the operations specified by the source code. Save as NGC (Native Generic Circuit) file.
  - For example: an expression with an addition operator (+) is interpreted as a binary adder with full-adder circuits. A statement with a condition operator such as

    ```
    assign Y = S ? In_1 : In_0;
    ```

    translates into a two-to-one-line multiplexer with control input S and data inputs In_1 and In_0.

FPGA Design Flow

- Implementation (three steps)
  - Translate
    - Combine all the input netlists and constraints to a logic design file, NGD Native Generic Database) file.
    - Assign the ports in the design to the physical elements (e.g., pins, switches, buttons) of the targets device and specifying time requirements of the design. Save as UCF (User Constraints File)
  - Map
    - Divides the whole circuit with logic elements into sub blocks (e.g., CLB) such that they can be fit into the FPGA logic blocks.
  - Place and Route
    - Place the sub blocks into logic blocks according to the constraints and connects the logic blocks.

FPGA Design Flow

- Device Programming
  - The design is finally converted to a format (a .bit file) so that the FPGA can accept it.
  - BITGEN program deals with the conversion. The file can then be loaded to the FPGA using a cable (e.g, JTAG).

FPGA Design Flow

- Design Verification
  - Behavioural Simulation (RTL Simulation)
    - This simulation is performed before synthesis process to verify RTL (behavioral) code and to confirm that the design is functioning as intended.
  - Functional Simulation (Post Translate Simulation)
    - Functional simulation gives information about the logic operation of the circuit. Designer can verify the functionality of the design using this process after the Translate process.
  - Static Timing Analysis
    - This can be done after MAP or PAR processes. Post MAP timing report lists signal path delays of the design derived from the design logic. Post Place and Route timing report incorporates timing delay information to provide a comprehensive timing summary of the design.