What is High-Level Synthesis?

◆ Many terms have been used:
  ▲ Algorithmic, Functional, Behavioral, Architectural, ...

◆ "High Level Synthesis means going from an algorithmic level specification of the behavior of a digital system to a register-level structure that implements that behavior."

▲ Input specification: gives required mappings from sequences of inputs to sequences of outputs.
▲ Must satisfy constraints: cycle-time, area, power, etc.
Algorithms and Programs

◆ Algorithm: specifies a sequence of steps which must be carried out in order to solve a problem
  ▲ Algorithm = Control-flow Program
  ▲ (Parallel Algorithm = Parallel Control-flow Program)

◆ Program: specifies a set of operations (essentially unordered) which must be carried out (in an appropriate order, if need be) on a set of input data in order to produce a desired set of output data.
  ▲ Needed order is a partial order defined by:
    ▼ Data dependencies
    ▼ Temporal dependencies (if sequential).

Sequential Control-Flow Model

◆ Based on Von Neumann "fetch;execute;store" with single (serial control flow) or synchronized multiple (parallel control flow) thread(s) of control.

◆ Proposes an ordering of computation based on strict, temporal sequencing of operations, in line with above cycle.

◆ Not well suited to the description of hardware since arbitrary hardware rarely fits such a model of computation.

◆ Ability to combine several operations and treat them as a unit, or block (begin-end, parbegin-parend, ...)

◆ Ability to allow an operation to be executed in a loop.

```c
j=0;
while(j < 10) {
  x[j]=y[j]+c;
  j=j+1;
}
for(i=0 to 9) {
  x[i]=y[i]+c; /* "forall" */
}
```

What is High-Level Synthesis?

◆ "... the system to be designed is usually represented at the algorithmic level by a programming language like Pascal or Ada, or by an HDL that is similar to a programming language...." (e.g. ISPS, DSL, MIMOLA, VHDL Behavior)

◆ "...most high-level synthesis systems use procedural languages... "
Representing the Synthesis Problem: The Conventional Approach

The transactional (system) level
The behavioral (architectural) level
The register level
The sequential level
The combinational level

Representing the Synthesis Problem: An Alternative Approach

Combinational Behavior
Store

Apply the "Microscope Test" to the Design
What is an Architecture?

◆ Amdahl, Blaauw, and Brooks, 1964, defined three interfaces:

▲ **Computer Architecture**: "The attributes of a computer as seen by a machine language programmer."

▲ **Implementation**: "Actual hardware structure, logic design, and datapath organization."

▲ **Realization**: "Encompasses the logic technologies, packaging, and interconnection."

◆ A description of the behavior of a system that is independent of its implementation.

▲ Isomorphic to its "interface specification" (Siewiorek, Bell, Newell, 1971)

▲ For example:

▼ Instruction set definition of a computer

▼ Z-domain description of a filter

▼ Handshaking protocol for a bus

◆ May guide implementation (contain 'hints' or 'pragmas')

▲ e.g. a particular specification may lead naturally to a serial or parallel implementation.
What is an Architecture?

- Example: Instruction set

```
Inst_A
Inst_B
Inst_C
...
Inst_C may not follow Inst_A

Architecture
```

Not architecture

---

Specification vs. Description

- **Specification**: Saying what I want; describes behavior in terms of results.
  - e.g. $\forall A \{ A[i,j] \leftarrow 0 \}$
- **Description**: Saying how to do it; describes behavior in terms of procedure or process.
  - e.g. for($i=0; i<N; i++$)
    for($j=0; j<M; j++$)
    $A[i][j] = 0$;
- We do not have specification languages for general-purpose digital design. For some special-purpose applications (e.g. DSP) we do.
High-Level Synthesis

- How does one measure the quality of a solution?
  - Number of control steps (clock cycles in synchronous single-clock implementations)
    - Chaining and overlapped operations.
  - Number of hardware functional units (e.g. ALU, adder, multipliers)
    - Combinational or pipelined
  - Number of registers
  - Number of MUX inputs
  - Total connections and "unique connections."

Converting Procedural Descriptions to a Dataflow-Oriented Representation

- Determine scope of variables and apply single-assignment in the scope
- Convert Complex data-structures into simple types
- Unroll loops with constant loop-counts (if appropriate)
- Perform simple syntactic optimizations:
  - Move operations out of loops where possible
  - Simplify complex expressions
  - Extract common sub-expressions

  e.g. CMUDA: Value-trace (VT)
  HAL: CDFG, YSC: YIF
Complications to Dataflow Analysis

◆ Unrestricted goto statements (jumps)
  ▲ Introduce overly-pessimistic dependencies during static analysis
  ▲ Values of variables references in the statements following a label depend on the assignments to variables in the code sections that can jump to the label.

◆ Global variables
  ▲ i.e. variables declared outside of any function and therefore able to be shared by all functions
  ▲ Last-use cannot be determined without examining the entire program.

Complications to Dataflow Analysis

◆ Static Variables
  ▲ i.e. variables local to a function that retain their values between calls to the function
  ▲ Create dependencies between the order of the calls to a function and, in effect, represent a communication path between all functions which call the function.

◆ Aliasing (often due to call-by-reference or call-by-name)
  ▲ i.e. one or more name used to represent the same variable, which can result in hidden dependencies.
  ▲ Several functions can be called with the name (or address) of the same object at the same time and multiple functions may try to modify the object concurrently. Unfortunately, this condition depends on run-time behavior.
Complications to Dataflow Analysis

- The Single-Assignment Rule
  ▲ Variables used as "scratchpad" temporaries (assigned a value then re-assigned another value within the same section of program) create false dependencies between the old value and the new value of the variable.
  ▲ Apply rule that a variable may only be written to once within a scope.
    ▼ A := A+1;
    ▼ next A := A+1

- Applicative or Functional Languages
  ▲ No goto’s, global or static variables, call-by-reference or aliasing
  ▲ Enforces the single-assignment rule
  e.g. Silage, Ella

Conventional Steps in High-Level Synthesis

- Scheduling: Assignment of operations to control steps
- Allocation: Determining the number and types of operators needed to implement behavior
- Mapping: Assigning operations to specific operators
  ▼ Type selection: which type of operator should be used? (e.g. add on ALU or adder)
  ▼ Instance mapping: actual mapping to specific hardware module
Components of a Conventional High-Level Synthesis Target

◆ Data-part Components:
  ▲ Functional Units (operators): ALU, mult., adders, etc.
  ▲ Registers, register files, or memory
  ▲ Interconnection hardware: Buses, MUXs, nets

◆ Controller:
  ▲ Symbolic STG for FSM (e.g. PLA-based)
  ▲ Microcode for microprogrammed controller

◆ Connected by:
  ▲ Control nets (from control to data part): e.g. enable/address storage, MUX select, bus tri-state control
  ▲ Condition nets (from data part to control): condition outputs like result of branch-test, overflow

Parallelism, Pipelining, and Graph Folding

◆ Differential Equation Example: \( y'' + 5zy' + 3y = 0 \)
  (Paulin, et. al. 1986)

\[
\begin{align*}
  lv &= '1'; \\
  \text{while (lv = '1')} \text{ loop} \\
  &\quad zn <= z + dz; \\
  &\quad un <= u - 5\cdot z\cdot u\cdot dz - 3\cdot y\cdot dz; \\
  &\quad yn <= y + u\cdot dz; \\
  &\quad lv <= (zn < a); \\
  &\quad z <= zn; \ u <= un; \ y <= yn; \\
  &\text{end loop;}
\end{align*}
\]
Maximally Parallel and Maximally Pipelined

```plaintext
lv = '1';
while (lv = '1') loop
    zn <= z + dz;
    un <= u - 5•z•u•dz - 3•y•dz;
    yn <= y + u•dz;
    lv <= (zn < a);
    z <= zn; u <= un; y <= yn;
end loop;
```

Maximally Parallel, Minimum Control States

```plaintext
Maximally Parallel, Minimum Control States
```

```plaintext
3 1
2 1 1
1
1
```
Parallelism, Pipelining, and Graph Folding

- **Parallelism**: Multiple functional units at the same level in the graph active at the same time.
- **Pipelining**: Functional units on multiple levels of the graph being active at the same time.
- **Widthwise Graph Folding**: restricts parallelism with the addition of symbolic dependencies.
  - Limiting case: Multi-stage pipelined processor with feedback and memories between the stages.
- **Lengthwise Graph Folding**: decreases the number of levels in the graph with the addition of symbolic multiplexers.
  - Similar to colored tokens in dataflow computer systems (Arvind, 1978).
  - Implementation of the folded graph need not use actual multiplexers.
  - Limiting Case: Parallel processor with an interconnection network to switch values between processors and memories.
Maximally Pipelined: Storage Requirements

Sharing Storage
Effect of Scheduling on Interconnects

"... ignores the effects ... that a schedule could have on the interconnect part of the design..."
Cloutier and Thomas, DAC 1990

"... the amount of storage and communication cannot be influenced significantly by the schedule ..."
Camposano, Trans. CAD, Jan 1991
(quoting L. Stok, Proc EDAC 90)

Approaches to Scheduling

(1) Exhaustive Approaches
e.g. EXPL (Barbacci, 1962): exhaustive search,
Hafer & Parker: Branch & bound

(2) As-Soon-As-Possible (ASAP)
Used by many pioneering systems (e.g. early CMUDA, MIMOLA, and Flamel)
As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) Scheduling

Approaches to Scheduling

(3) List Scheduling
- Keep list of operations available to be scheduled (i.e. whose predecessors have already been scheduled) and order them for scheduling using some priority function
  - BUD - length of path from operation to end of enclosing block
  - Elf, ISYN - "urgency": length of shortest path from operation to nearest local constraint
- O(C NlogN) time complexity

(4) Force-Directed Approaches
- "Force" between operation and control-step proportional to the number of operations of the same type that could go in that control step
- Scheduling to minimize force tends to minimize the number of resources needed (e.g. HAL)
- O(C N^2) time complexity