High-Level Synthesis

a.k.a. Behavioral Synthesis

a.k.a. Algorithm Level Synthesis

a.k.a. Silicon Compilation

- ***High-Level Synthesis*** **(HLS) takes a specification of the functionality of a digital system and a set of constraints, finds a structure that implements the intended behavior, and satisfies constraints**
- **The basic elements of algorithmic description correspond to the basic elements of programming or hardware description languages. They comprise arithmetic and logic operation, variables or values, and control constructs like if/case, loops and procedure calls.**
  - **Textual entry (synthesizable subset)**
  - **Graphical entry (e.g., flow-chart)**

Benefits

- **Automatization simplifies handling of larger designs and speeds up exploration of different architectural solutions.**
- **The use of synthesis techniques promises correctness-by-construction. This both eliminates human errors and shortens the design time.**
- **The use of higher abstraction level, i.e. the algorithmic level, helps the designer to cope with the complexity.**
- **An algorithm does not specify the structure to implement it, thus allowing the HLS tools to explore the design space.**
- **The lack of low level implementation details allows easier re-targeting and reuse of existing specifications.**
- **Specifications at higher level of abstraction are easier to understand thus simplifying maintenance.**

Example – Differential Equation

$$\frac{d^2y}{dx^2} + 5\frac{dy}{dx}x + 3y = 0$$

```
  variable a,dx,x,u,y,x1,y1: integer;
begin
  cycles(sysclock,1); a:=inport;
  cycles(sysclock,1); dx:=inport;
  cycles(sysclock,1); y:=inport;
  cycles(sysclock,1); x:=inport;
  cycles(sysclock,1); u:=inport;
  loop
    cycles(sysclock,7);
    x1 := x + dx; y1 := y + (u * dx);
    u := u-5 * x * (u * dx) - 3 * y * dx;
    x := x1; y := y1;
    exit when not (x1 < a);
  end loop;
```


SW Compilation

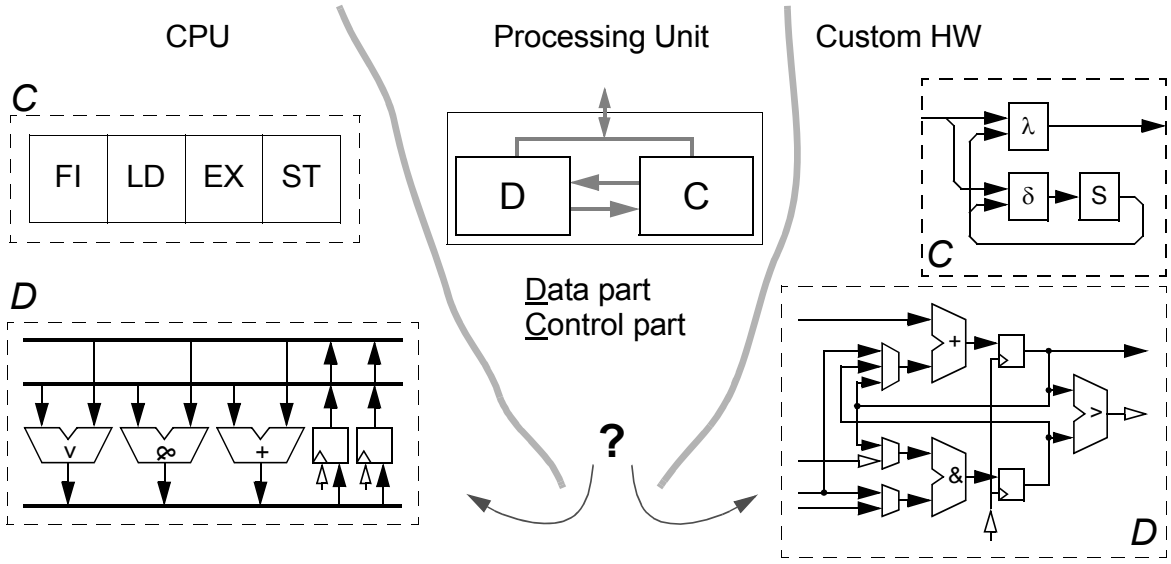
- **Example – differential equation** $\frac{d^2y}{dx^2} + 5\frac{dy}{dx}x + 3y = 0$

```
{
  sc_fixed<6,10> a,dx,y,x,u,x1,x2,y1;
  while ( true ) {
    wait(); a=inport.read();
    wait(); dx=inport.read();
    wait(); y=inport.read();
    wait(); x=inport.read();
    wait(); u=inport.read();
    while ( true ) {
      for (int i=0;i<7;i++) wait();
      x1 = x + dx;  y1 = y + (u*dx);
      u = u - 5*x*(u*dx) - 3*y*dx;
      x = x1; y = y1;
      if (!(x1<a)) break;
    }
    outport.write(y);
  };
}
```

```
# R1:a, R2:dx, R3:y, R4:x, R5:u,
# R6:x1, R7:x2, R8:y1, R9:tmp
    ...
_loop_$32:
    ADD.fx   R6, R4, R2     # x1=x+dx
    MUL.fx   R9, R5, R2     # tmp=u*dx
    ADD.fx   R8, R3, R9     # y1=y+tmp
    MUL.fx   R9, R4, R9     # tmp=x*tmp
    MUL.fx   R9, R9, $5     # tmp=5*tmp
    SUB.fx   R5, R5, R9     # u=u-tmp
    MUL.fx   R9, R3, R2     # tmp=y*dx
    MUL.fx   R9, R9, $3     # tmp=3*tmp
    SUB.fx   R5, R5, R9     # u=u-tmp
    ADD.fx   R4, R6, $0     # x=x1
    ADD.fx   R3, R8, $0     # y=y1
    SUB.fx   R9, R6, R1     # tmp=x1-a
    JMP.neg  _loop_$32      # ...break
    ...
```

Software vs. Hardware

The Target Architecture

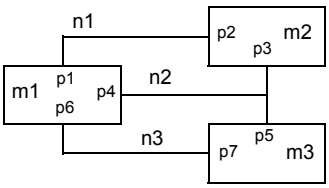


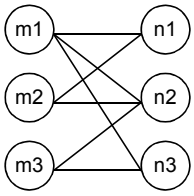
Target

- **SW synthesis (compilation)**
  - **input – high-level programming language**
  - **output – sequence of operations (assembler code)**
- **HW synthesis (HLS)**
  - **input – hardware description language**
  - **output – sequence of operations (microprogram)**
  - **output – RTL description of a digital synchronous system (i.e., processor)**
    - **data part & control part**
    - **communication via flags and control signals**
    - **discrete time steps (for non-pipelined designs *time step* = *control step*)**
- **Creating the RTL structure means mapping the data and control flow in two dimensions – time and area**

Target Representation

- The target architecture of High-Level Synthesis consists of data part netlist and controller part symbolic state transition table/graph
- The netlist G(C,N,E) is a bipartite graph in which the vertices C correspond to instances of HW components, the vertices N correspond to buses and nets, and the edges E correspond to terminals of components
- FSM(S,X,Y,f,g)
  - S - set of states
  - X - set of inputs
  - Y - set of outputs
  - f - transition function
  - g - output function



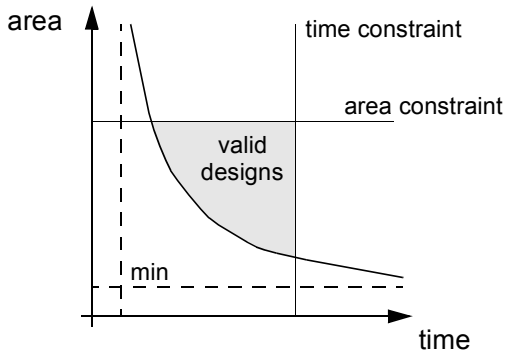


Data Part Interconnection Schemes

- Multiplexed bus architecture
  - storage is provided by distributed registers
  - interconnection is established by multiplexors and nets
  - single system clock (single edge)
- Bidirectional bus architecture
  - functional units have storage capabilities;
    either inputs or outputs are latched or associated with register
  - register files
  - interconnection is established by bidirectional busses
  - a two-phase clocking scheme may be needed.

Design Space

- The *main trade-off* is the serial/parallel trade-off
  area-efficient and slow versus area-intensive and fast designs
- Additional parameters:
  - performance
    - delay & cycle-time
    - latency
    - throughput
  - power consumption
  - testability
  - etc.



- Area is important even in submicron era!
  - yield in chip production
  - package cost

The Classical High-Level Synthesis Tasks

- Front-end:
  - Deriving an internal graph-based representation equivalent to the algorithmic description of both the data flow and the control flow
  - Compiler optimizations
- Back-end:
  - Behavioral transformations (control and/or data graph transformations e.g. associativity, unrolling)
  - Transforming data and control flow into register-transfer level structure (so called *essential subtasks*)
  - Netlist extraction, state machine table generation

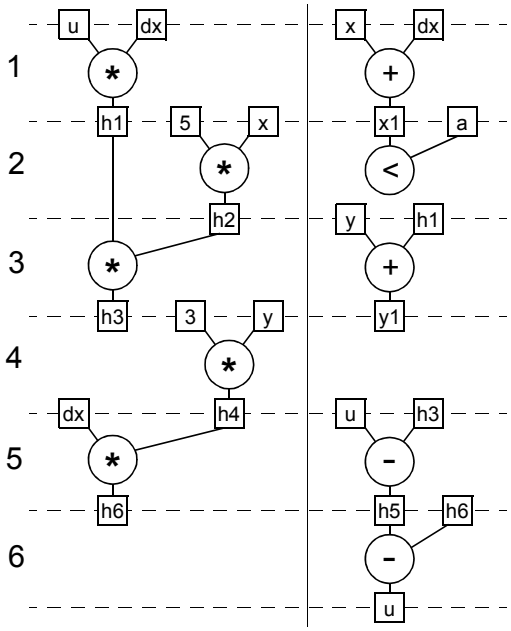
Essential Subtasks

- Scheduling
  - Assignment of operations to time steps subject to certain constraints and minimizing some objective function
- Resource allocation
  - Number and types of functional units
  - Number and type of storage elements
  - Number and type of busses
- Resource assignment
  - Operations to functional unit instances
  - Values to be stored to instances of storage elements
  - Data transfers to bus instances

Subtasks – Another View

- Partitioning
  - Divides a behavioral description into sub-descriptions in order to reduce the size of the problem or to satisfy some external constraints.
- Allocation
  - The task of assigning operations onto available functional unit types available in the target technology.
- Scheduling
  - The problem of assigning operations to control steps in order to minimize the amount of used hardware. If performed before allocation (and binding), it imposes additional constraints how the operations can be allocated.
- Binding
  - Assigns operations to specific instances of unit types in order to minimize the interconnection cost.

Minimal Hardware



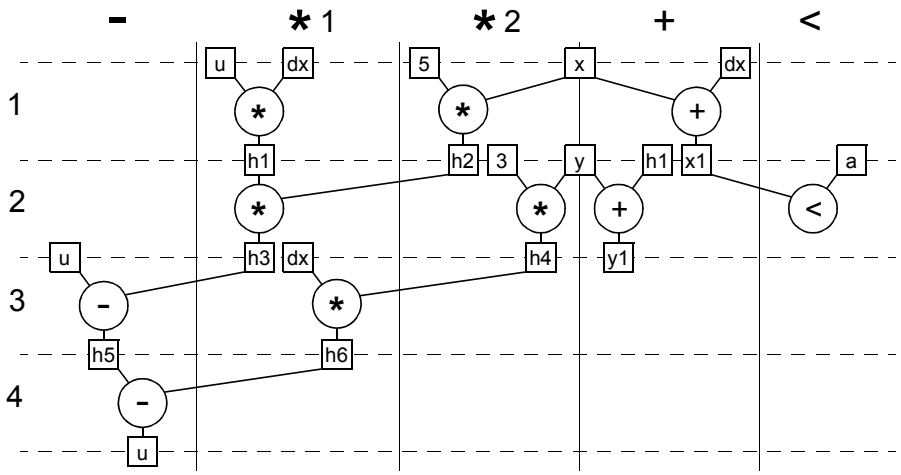
$$\frac{d^2y}{dx^2} + 5\frac{dy}{dx}x + 3y = 0$$

* FU1

+/−/< FU2

Minimal Time

$$\frac{d^2y}{dx^2} + 5\frac{dy}{dx}x + 3y = 0$$



Practicality of the High-Level Synthesis

- Data dominated circuits
  - Characterized by intensive arithmetic computation. Parallelization is key to meet throughput constraints as well pipelining.
  - Used in digital signal processing (DSP) applications.
  - Often combined with complex memory management units.
- Control dominated circuits
  - Typically do not require many arithmetic operations nor do many operations have to be executed in parallel. Extensive usage of procedural control constructs is typical.
  - The typical target is multiplexed bus architecture.
- Memory intensive circuits

Internal Representation of the Algorithmic Description

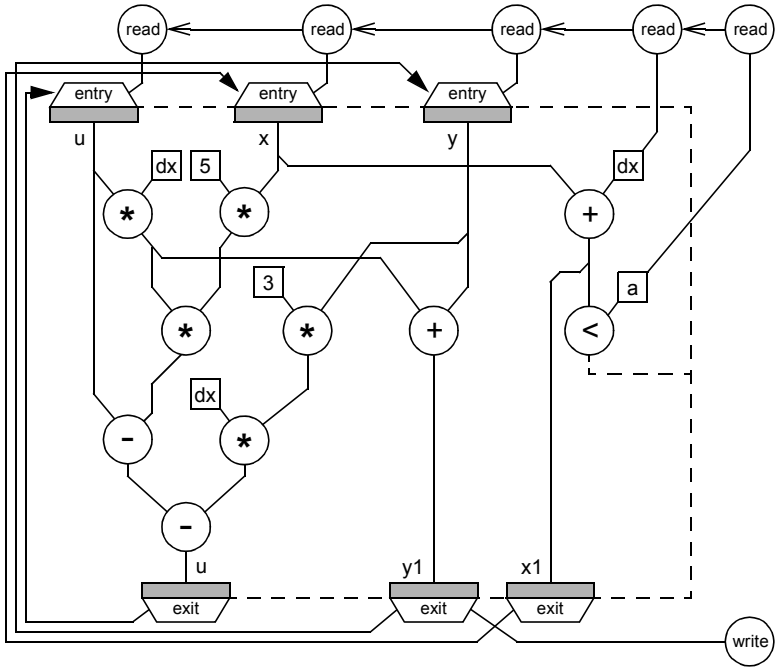
- Control flow model – CFG(V,E)
  - nodes – basic blocks
  - edges – flow of control
- Data flow model – DFG(V,E)
  - nodes – actors, representing operations
  - edges – links, representing data conveying paths
  - DFG specifies a partial order of operations
- Control flow and data flow can be combined in a single graph – CDFG (Control and Data Flow Graph)

Eindhoven Silicon Compiler

DFG(V_o,V_c,E,E_c,E_s)

- **Data flow edges**
- **Control edges**
- **Sequence edges**
- **Operation vertices** (+)
- **Read and write vertices** (read)
- **Constant and variable vertices** (5, a)
- **Branch/merge vertices** (branch, merge)
- **Loop entry/exit vertices** (exit, entry)

Differential Equation Example



Synthesis

- **Synthesizing an appropriate RT level structure implies meeting hardware constraints such as area, clocking frequency, delay, power consumption, etc.
Physical parameters, however, can be estimated from the physical parameters of the hardware components in the library.**

Component	Delay	Area
ALU(+,-,<)	24 ns	208
Adder	18 ns	125
Subtracter	19 ns	139
Comparator (<)	16 ns	72
Parallel Multiplier	49 ns	2284
2:1 Multiplexer || Tristate driver	2 ns	48
Register	1 ns	112
2-stage Pipelined Multiplier	28 ns	2624

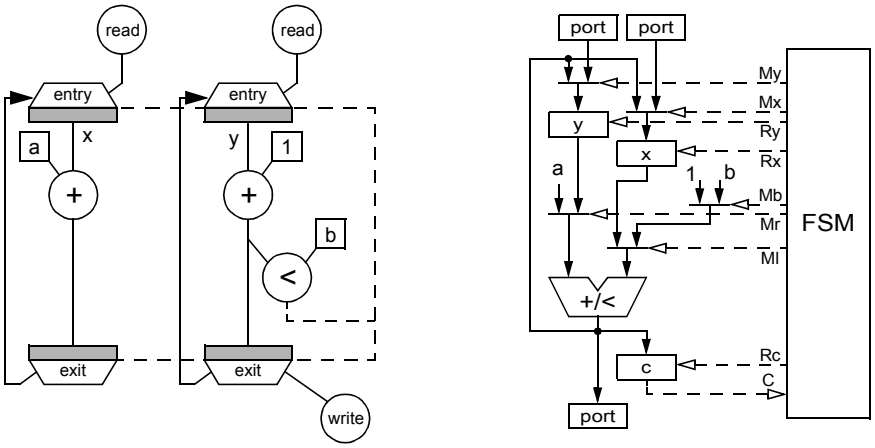
example parameters - LSI-10K, 16-bit units

Scheduling Decisions

- **Chip area estimation - on the basis of selected data part architecture.**
- **Time is abstracted to the number of needed time steps.
Time is estimated using the hardware component delays and rough estimates of the contributions by storage and interconnects.**
- **Depending on whether the time constraint or the area constraint is more difficult to meet, *resource constrained scheduling* or *time constrained scheduling* has to be chosen.**

Synthesized Structure

- Internal representation: netlist G(C,N,E) and FSM(S,X,Y,f,g)



High-Level Synthesis in Practice – #1

- Traffic Light Controller (TLC)
  - Manual transformation from simulatable specification to synthesizable code
  - Step-by-step refinements of the original code
- From idea to model
  - Simulatable specification – behavioral VHDL
  - Validation environment – test-bench
- From model to structure
  - abstract data-types → bit-level data-types
  - complex time control operations → synchronization + counter (& timer)
  - introducing structure → controller, timer, and logic for blinking
- From structure to schematics
  - Synthesizable code at Register-Transfer Level

High-Level Synthesis in Practice – #2

- 9-tap Finite Impulse Response (FIR) filter
  - Behavioral description is not synthesizable!
  - Multiplication is too slow!
  - Naive synthesizability (Behavioral RTL) is too expensive!
- Code transformations
  - multiplications → shift-add trees
- Allocation
  - adders, subtracters, and/or adder-subtracters?
- Time constrained scheduling
  - 9 multiplications & 8 additions → ~12 additions/subtractions – all in four clock-steps!
- Binding
  - adders/subtracters, multiplexers, and registers

Conclusion

- *General* or *domain specific* synthesis?
  - Processor applications
  - Control oriented systems
  - Video processing
  - DSP applications
  - Telecommunication
- Specific subtasks:
  - Memory management unit synthesis
  - Control partitioning, partial encoding of states, encoding of control signals.
  - High-level data part mapping