

ECE 587 – Hardware/Software Co-Design

Lecture 23 Hardware Synthesis I

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

April 6, 2026

Reading Assignment

- ▶ This lecture: 6
- ▶ Next lecture: 6

Hardware Synthesis

Simplified HLS Flow

Hardware Synthesis via High-Level Synthesis (HLS)

- ▶ A method to generate hardware implementations from behavioral descriptions
- ▶ Input
 - ▶ Behavioral descriptions, e.g CDFG
 - ▶ RTL library: available hardware resources
 - ▶ Design constraints
- ▶ Output: RTL netlist
 - ▶ A synchronous circuit consisting of functional units, registers, interconnects, and control logics.
- ▶ From untimed behavior to cycle-accurate behavior

- ▶ Although the output of HW synthesis is the RTL/FSM model of the component, we can divide it into pieces.
 - ▶ Facilitate reasoning/communications
 - ▶ Optimize with specialized algorithms
- ▶ Datapath
 - ▶ Perform complicated, but usually combinational, computations
 - ▶ Produce status signals for decision making
- ▶ Controller
 - ▶ Provide control signals to the datapath, e.g. to collect input data and to distribute output data
 - ▶ Interact with other components, e.g. to notify completion and to start computation once activated

Controller vs. Datapath

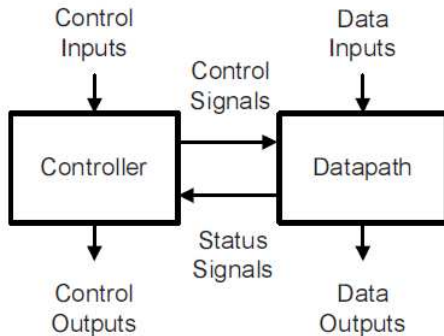


FIGURE 6.2 High-level block diagram

(Gajski et al.)

Controller vs. Datapath: A Detailed Diagram

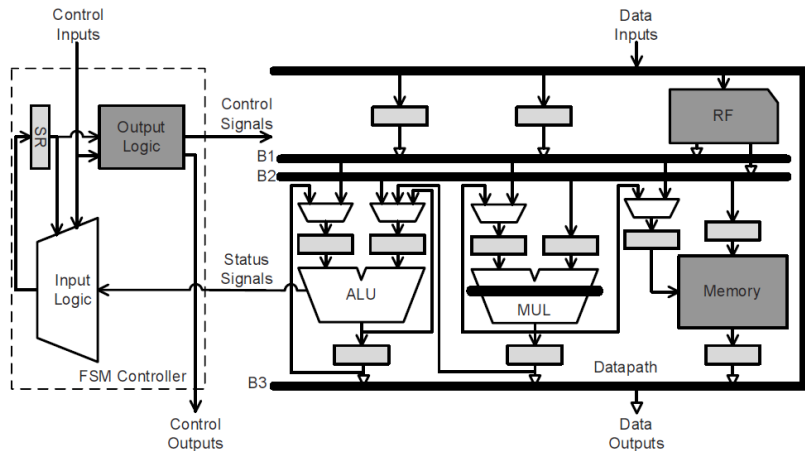


FIGURE 6.3 RTL diagram with FSM controller

(Gajski et al.)

Controller and Datapath Implementations

- ▶ The FSM of the whole HW component can be decomposed into a FSM.
 - ▶ The controller as a FSM (much less states compared to the FSM for the whole component)
 - ▶ The datapath as data and their operations associated with each state transition
- ▶ Controller FSM
 - ▶ Current state: stored in State Register (SR)
 - ▶ State transition: computed via input logic
 - ▶ Inputs: including original inputs and status bits returned by datapath operations
 - ▶ Outputs: computed via output logic, including original outputs and signals to activate corresponding datapath operations
- ▶ Datapath
 - ▶ Registers: for data storage and pipelining
 - ▶ Functional units: for computation
 - ▶ Interconnects: shared busses and wires, plus muxes and tri-state buffers for multiplexing

HW Synthesis Design Flow

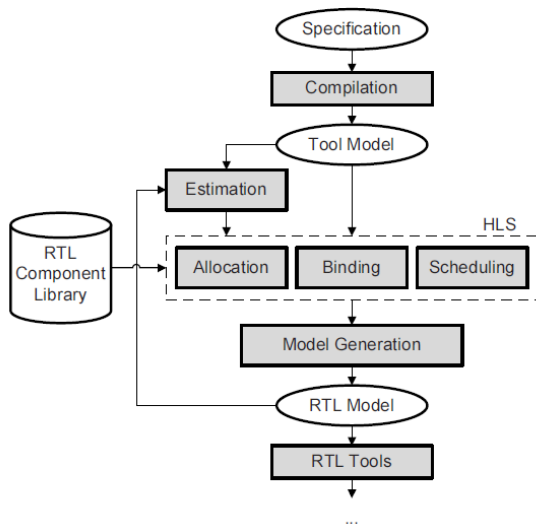


FIGURE 6.1 HW synthesis design flow

(Gajski et al.)

HLS Tasks

- ▶ Allocation: spatial aspects of the hardware system
 - ▶ Determine the type and quantity of hardware resources used
 - ▶ Directly affect chip area
- ▶ Scheduling: temporal aspects of the hardware system
 - ▶ Determine the clock period
 - ▶ Determine when (the clock cycle) the activities in CDFG should be executed
 - ▶ Directly affect total execution time
- ▶ Binding: connecting spatial and temporal aspects
 - ▶ Determine the hardware resources to execute each activity
 - ▶ Become important as metrics beyond chip area and execution time are of concern
- ▶ An optimal HLS algorithm should explore all possibilities in these tasks
 - ▶ A very difficult problem.

Possible HLS Flows

- ▶ Complete allocation/binding/scheduling sequentially
- ▶ Pre-allocation: define architecture for HW processor first
- ▶ Pre-binding: optimize register usage first
- ▶ Pre-scheduling: avoid structural dependency in inner loops first

Hardware Synthesis

Simplified HLS Flow

Simplified HLS Setting

- ▶ Input
 - ▶ Behavioral description based on DFG
 - ▶ Ignore design constraints
- ▶ Fixed allocation for functional units
 - ▶ Functional units are combinational: no pipelining, need storage units for input/output
- ▶ Scheduling
 - ▶ Fixed clock period
 - ▶ No *chaining*: no data-dependency among activities executed in one clock cycle
- ▶ Objective: generate a hardware implementation with reasonably good performance and cost

Simplified Design Flow

1. DFG generation
 2. Scheduling and functional units binding
 3. Storage units allocation and binding
 4. Control unit synthesis
- ▶ 2 and 3 are usually known as datapath synthesis

Example Program

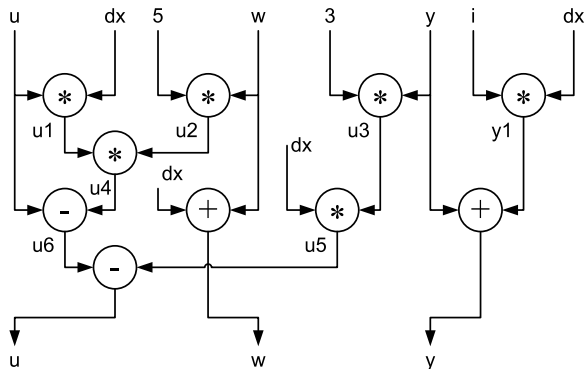
```
double u, w, y, dx;
int i, N;

for (i = 0; i < N; ++i) {
    double u1, u2, u3, u4, u5, u6, y1;

    u1 = u *dx;
    u2 = 5 *w;
    u3 = 3 *y;
    y1 = i *dx;
    w = w +dx;
    u4 = u1*u2;
    u5 = dx*u3;
    y = y +y1;
    u6 = u -u4;
    u = u6-u5;
}
```

- ▶ Assume we need to speed-up the loop body by hardware implementations

Step 1: DFG Generation



- ▶ Based on data dependency of loop body
- ▶ Vertices are operations (activities), edges are variables

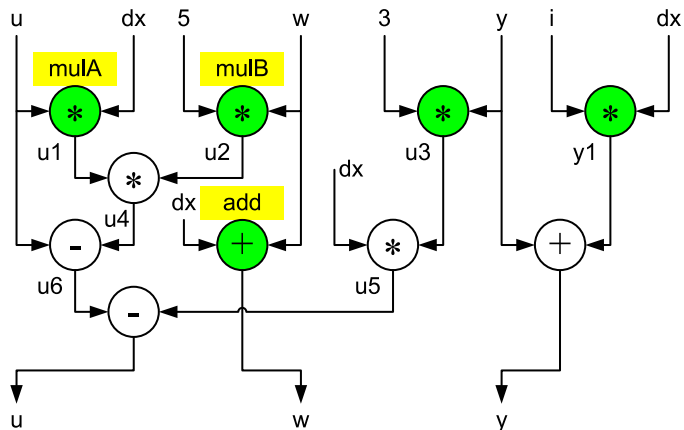
Example Functional Units Allocation

- ▶ add: need 1 clock cycle to generate result
- ▶ sub: need 1 clock cycle to generate result
- ▶ mulA and mulB: need 4 clock cycles to generate result

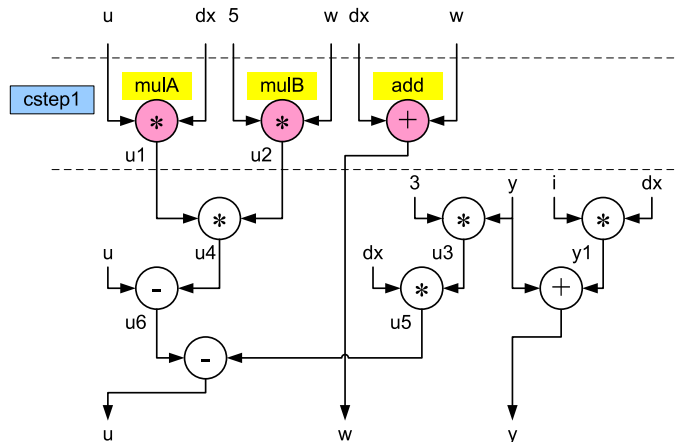
Step 2: Scheduling and Functional Units Binding

- ▶ *Control step* (cstep): usually equivalent to a clock cycle
 - ▶ Corresponds to the lifetime of a single state in the FSM representing the control unit
- ▶ Scheduling and functional units binding algorithm
 - ▶ Assign operations to functional units iteratively until all operations are assigned
 - ▶ Assume external variables to loop body (e.g. u , w , y , i , dx) are ready in cstep 0 and scheduling starts in cstep 1
 - ▶ Each iteration handles one cstep
 - ▶ Data dependency: only operations with no unfinished predecessor at the beginning of the cstep can start execution in the cstep
 - ▶ Structural dependency: subject to the availability of functional units

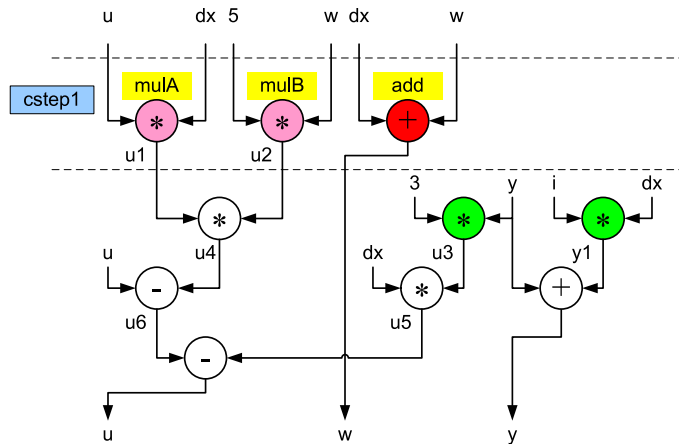
Control Step 1: Beginning



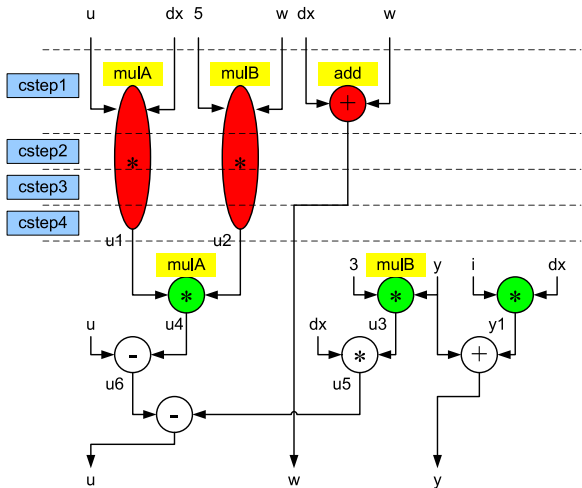
Control Step 1: Ending



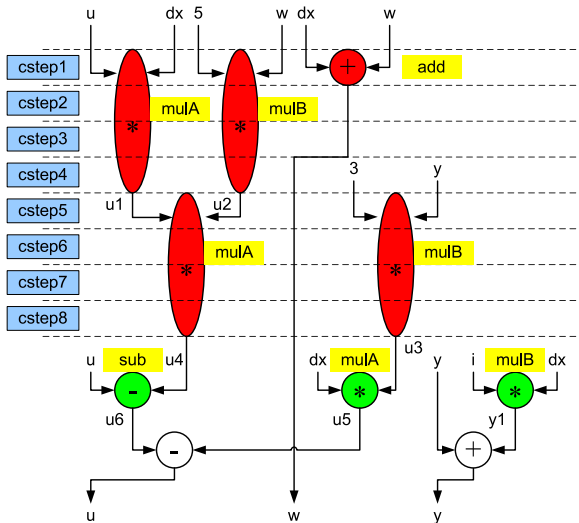
Control Step 2: Beginning



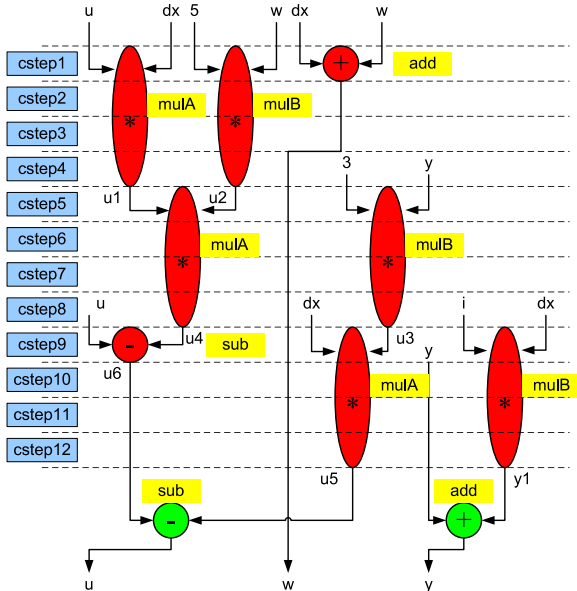
Control Step 5: Beginning



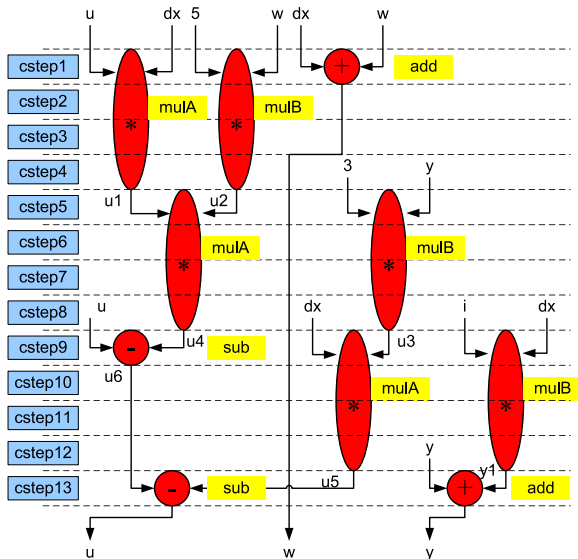
Control Step 9: Beginning



Control Step 13: Beginning



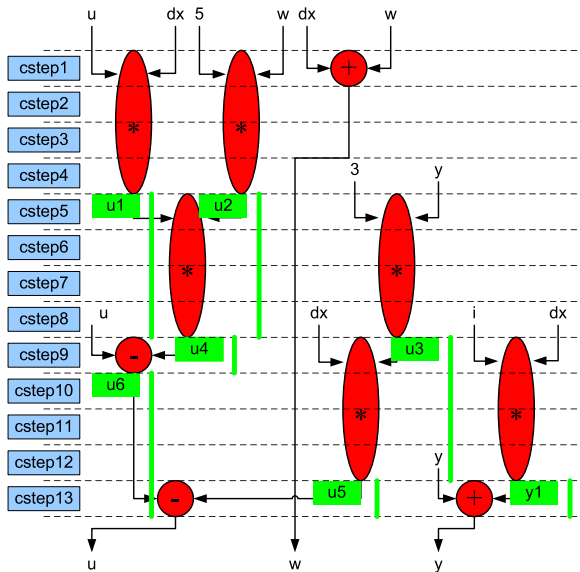
Overall Scheduling and Functional Units Binding



Step 3: Storage Units Allocation and Binding

- ▶ Storage units: registers (flip-flops)
- ▶ The straight-forward approach: allocate a register to each variable
 - ▶ Drawbacks: may need more than necessary number of registers, increase chip area
- ▶ Solution: share registers among variables
 - ▶ Variable *lifetime*: time interval between its definition to its last use
 - ▶ Two variables can share a register if their lifetimes don't overlap
- ▶ Assume external variables to loop body (e.g. u , w , y , i , dx) won't share registers with other variables

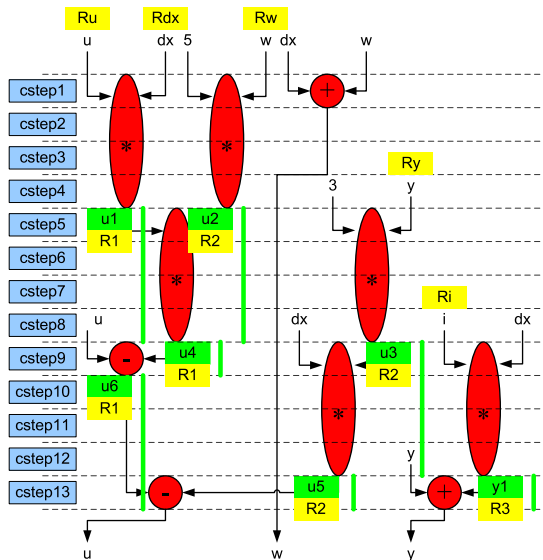
Variable Lifetimes



The Left Edge Algorithm for Register Allocation and Binding

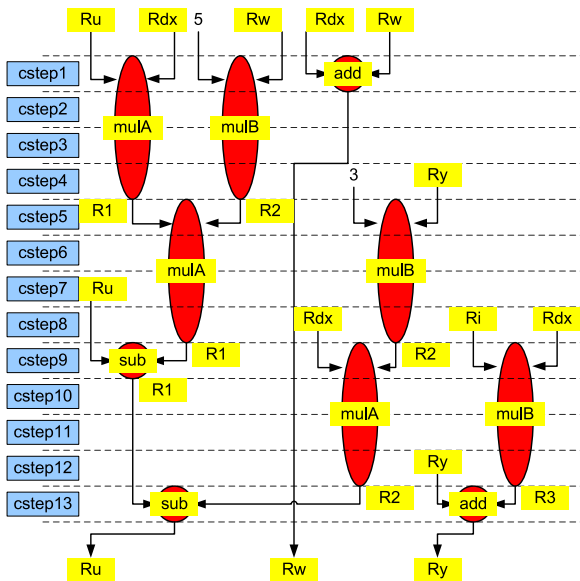
1. Sort variables into a list by the starting points of their lifetimes in ascending order
2. Allocate a register R for the first variable and remove the variable from the list
3. Bind R to the first variable in the list such that there is no overlap of lifetimes and remove the variable from the list
4. Repeat 3 until no such variable exists
5. Repeat 2 to 4 until there is no variable in the list

Register Allocations

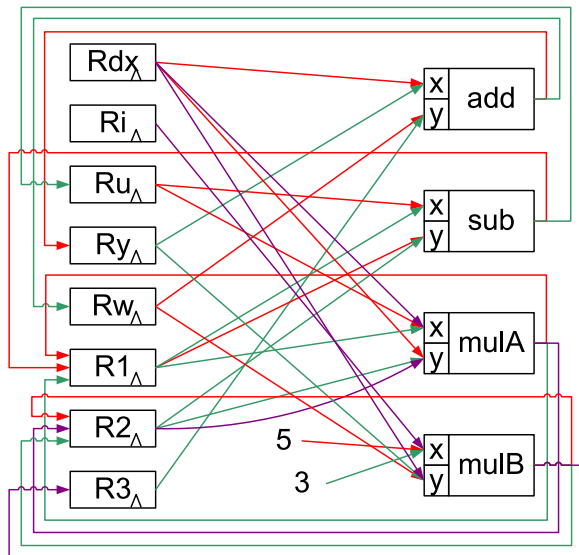


- ▶ Need $5+3=8$ instead of $5+7=12$ registers

Overall Scheduling and Binding



Hardware Diagram



Step 4: Control Unit Synthesis

- ▶ The Diagram in the previous slide is clearly not completed
 - ▶ An input port cannot be driven by multiple signals
 - ▶ A register should hold its data until being explicitly changed
- ▶ Use a mux at each input to choose the correct signal
- ▶ Control unit synthesis: design FSMs to generate the control signals for the muxes
 - ▶ State transition depends on scheduling and binding

Generate State Transitions for FSM Design

Port	csteps												
	1	2	3	4	5	6	7	8	9	10	11	12	13
mulA.x	Ru	Ru	Ru	Ru	R1	R1	R1	R1	Rdx	Rdx	Rdx	Rdx	*
mulA.y	Rdx	Rdx	Rdx	Rdx	R2	R2	R2	R2	R2	R2	R2	R2	*
Ry	Ry	Ry	Ry	Ry	Ry	Ry	Ry	Ry	Ry	Ry	Ry	Ry	add
R1	*	*	*	mulA	R1	R1	R1	mulA	sub	R1	R1	R1	*
R2	*	*	*	mulB	R2	R2	R2	mulB	R2	R2	R2	mulA	*

Summary

- ▶ Hardware synthesis is based on high-level synthesis that converts behavior specification into RTL.
- ▶ Dependencies among allocation/binding/scheduling make HLS difficult.
 - ▶ Functional units allocation affects scheduling and function units binding.
 - ▶ Scheduling and function units binding affects storage units allocation and binding.
 - ▶ Scheduling and binding affects control unit design.