
EEL 4783: Hardware/Software Co-design with FPGAs

Lecture 11: Hardwar/Software Partitioning *

*

Prof. Mingjie Lin

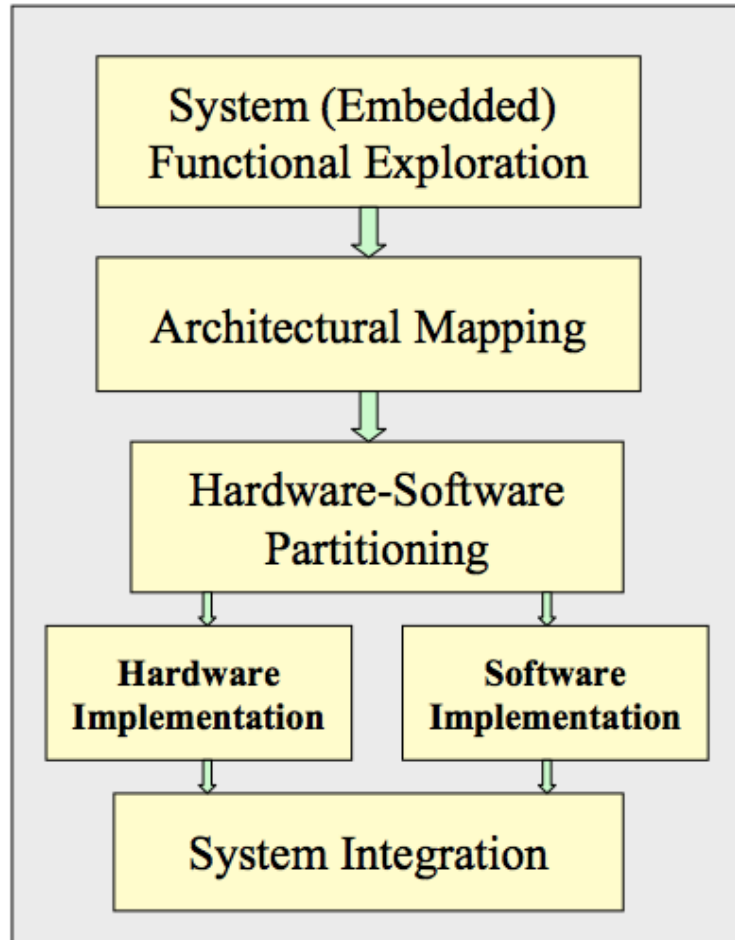


* Adopted from G. Khan COE718:

Embedded Computing System

- HW and SW used to be designed separately
- Co-Design an increasingly important object, because
 - Performance
 - Cost
 - Complexity
 - ?

HW/SW Co-Design



- Functional exploration: Define a desired product's requirements and produce a specification of the system behavior.
- Map this specification
- Partition the functions between silicon and code, and map them
- Integrate system

HW/SW Co-Design

- Co-Specification: Describe system functionality at the abstract level
- System description is converted into a task graph representation
- HW-SW Partitioning: Take the task graph and decide which components are implemented where/how ?
 - i.e. Dedicated hardware Software

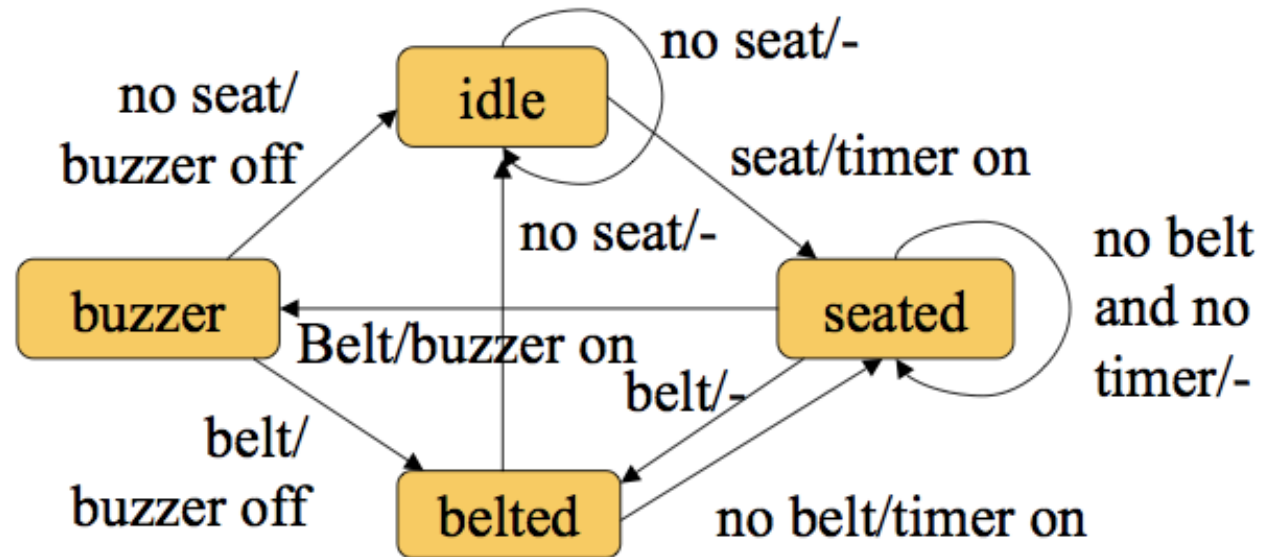
HW/SW Co-Design

- Both textual and graphical representation like DAG are used to describe system.
- Analyzes task graph to determine each task's placement
- Many partitioning algorithms being developed
- Major problem involves the computation time of the algorithm.

System Design Patterns

- Design Pattern: A generalized description of the design of a certain type of program that can also be used for system representation and hardware-software partitioning.
- State Diagram
Data Flow Graph
Control Data Flow Graph (CDFG)
 - Others
 - Directed Acyclic Graph (DAG) similar to DFG
 - Directed Acyclic Data Dependence Graph with Precedence (DADGP)

State Machine: Seat-belt System



```
switch (state) {  
    case IDLE: if (seat) { state = SEATED; timer_on = TRUE; } break;  
    case SEATED: if (belt) state = BELTED;  
                 else if (timer) state = BUZZER; break;  
    .....  
}
```

Data Flow Graph (DFG)

- DFG does not represent control
- It models the Basic Block: code or a system block with one entry and exit
- Describes the minimal ordering requirements on operations

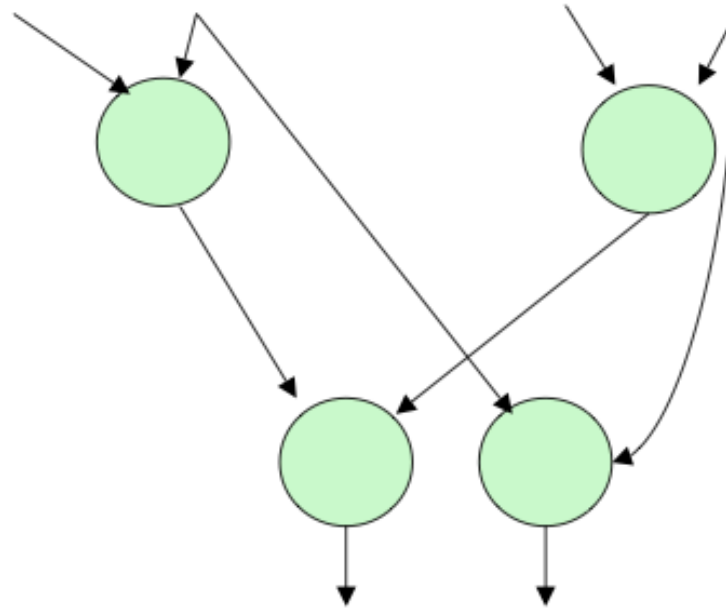
Data Flow Graph: Software Module

$x = a + b;$

$y = c - d;$

$z = x * y;$

$y1 = b + d;$



DFG

Control Data Flow Graph

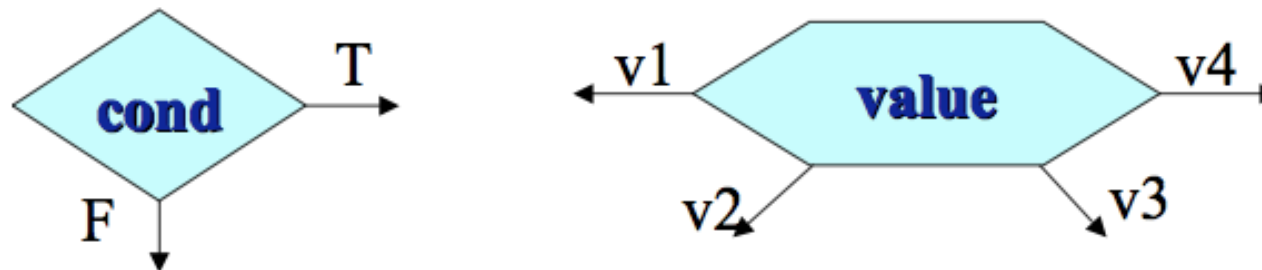
CDFG: represents control and data.

- Uses data flow graphs as components.
- Two types of nodes:

- **Data Flow Node encapsulate a DFG**

```
x = a + b;  
y = c + d
```

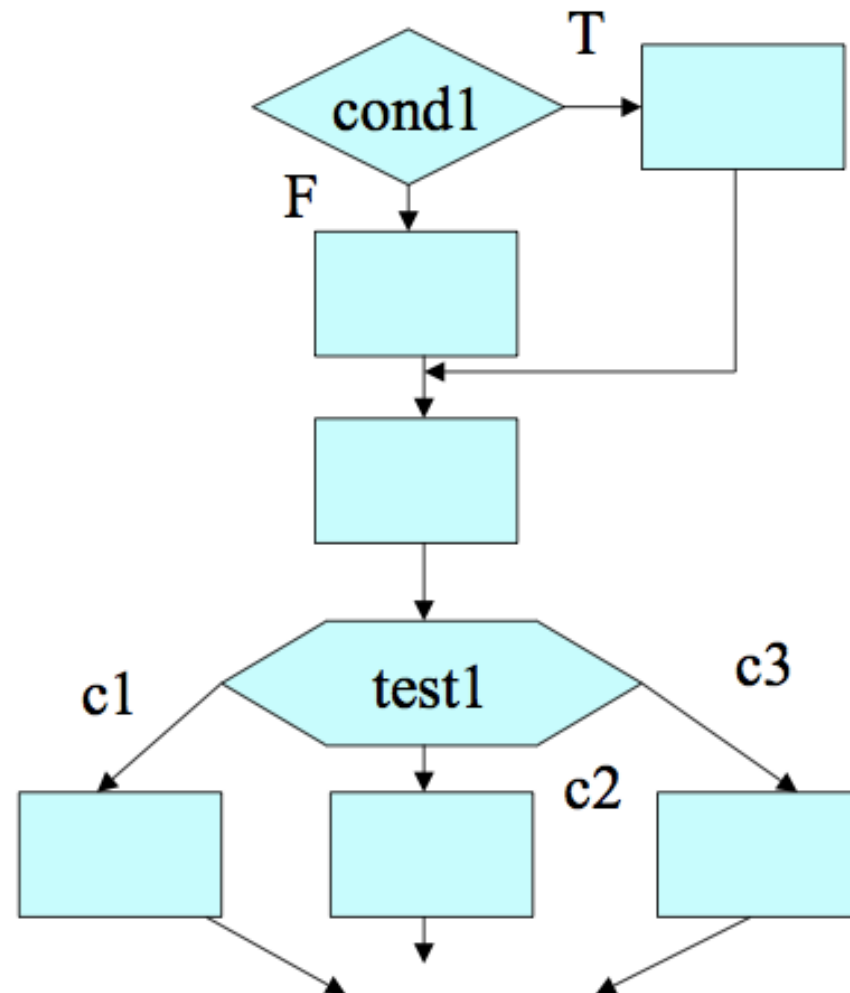
- **Decision Nodes**



Equivalent Forms

Control Data Flow Graph Example

```
if (cond1) bb1();  
else bb2();  
bb3();  
switch (test1) {  
  case c1: bb4(); break;  
  case c2: bb5(); break;  
  case c3: bb6(); break;  
}
```



Scheduling and Partitioning

- The main input to scheduling for partitioning is a graph representation in the form of DFG.
- Complex designs contain thousands of both control and data processing operations ranging from:
 - Complex arithmetic operations or logic-level bit- operations.
 - All the above interleaved operations by multiple control operations and loops.
- Such designs contain thousands of data-dependencies, basic blocks and control paths.

DFG-based Scheduling & Partitioning

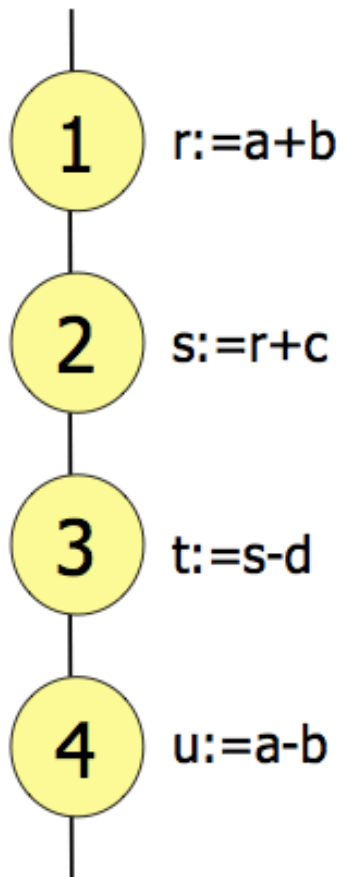
- Data-flow based scheduling techniques extract parallelism from the input description
- Schedule operations in parallel to satisfy the constraints.
- Two most common DF-based scheduling methods.
 - List Scheduling (LS): Minimize the number of control steps under resource constraints.
 - Force-directed Scheduling (FDS): Minimize the number of resource constraints under a fixed number of control steps.

Data Flow: DF-Scheduling

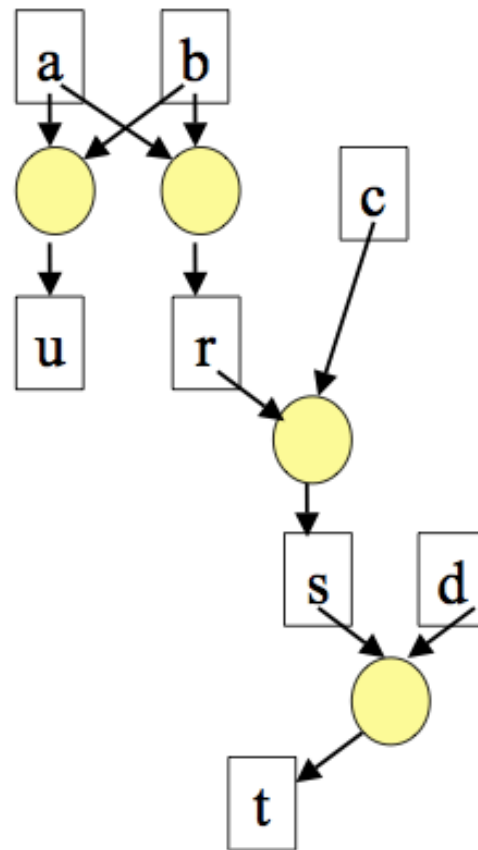
- List scheduling algorithm uses a cost function to select the operation to be scheduled from a list.
- DF-approach provides flexible cost function and it can be easily adapted to generate resource-constraint as well as time-constraint schedules.
- The cost function can represent any design measure such as HW area, delay, etc.
- The result is only as good as the cost function.
- DF-based algorithms can analyze all the parallelism in the DFG independently.

DF- Scheduling Example

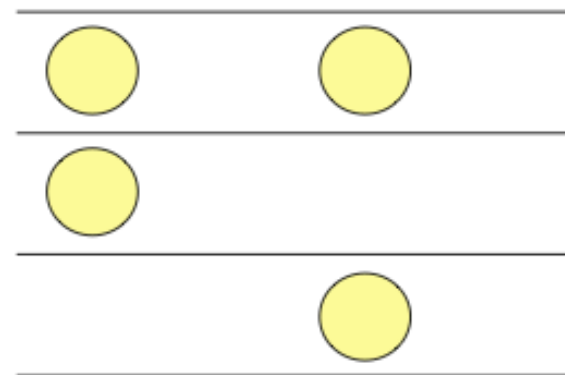
CFG



DFG



DFG-Schedule



Control Flow: CF-Scheduling

- Analyze the sequences of operations in CFG called control flow paths and schedule the CFG with minimum number of control steps in each path.
- Path-based scheduling is one of the main example of this scheme.
- Analyze all the paths in the CFG and schedule each of them independently.
- It minimizes the number of control steps in each path rather than minimizing the number of states.
- Paths in CFG come from loops and conditional operations.

Partitioning Approaches

- Simple one CPU and single ASIC architecture is the most common.
 - Early approaches Initially assume all tasks mapped to software
 - Move tasks to HW incrementally until system requirements are met.
 - Other early approaches: Initially all tasks are mapped to dedicated hardware.
 - Move tasks incrementally to SW until system requirements are met.
-

Optimal Partitioning

- Exhaustive approaches are characterized by attempting all possible combinations there by always selecting the best option.
- Exhaustive approaches are generally computationally intensive, consume huge-time in the range of hours or even days to find an optimal partition.
- Limited to small task graphs
 - Large telecom or other embedded systems can have upto 4000 nodes

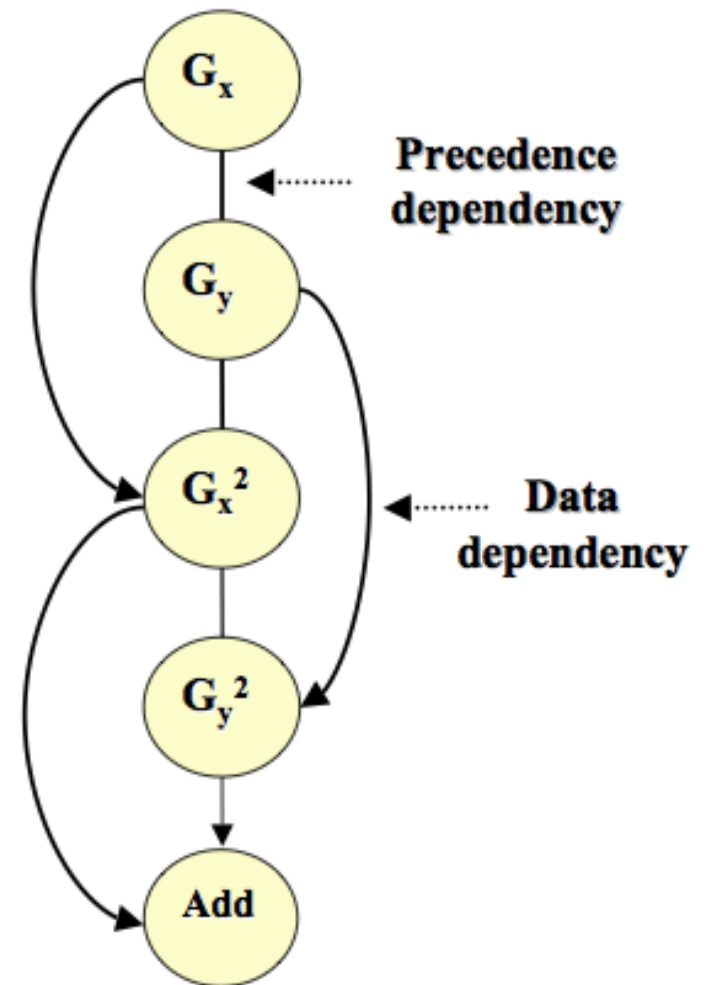
Edge Detection Example

Pair of masks are convolved to estimate gradients, G_x and G_y

$$\text{Overall } G^2 = (G_x^2 + G_y^2)$$

HW-SW Library

Operation	SW EXE (ms)	HW EXE (ms)	HW Area (gates)
Gradient (G_x or G_y)	9.4	1.4	1200
Square	5.2	0.9	500
Add	3.88	0.3	100



SOBEL Edge Detection

SOBEL masks

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Input Image

a ₁₁	a ₁₂	a ₁₃		
a ₂₁	a ₂₂	a ₂₃		
a ₃₁	a ₃₂	a ₃₃		

Mask

m ₁₁	m ₁₂	m ₁₃
m ₂₁	m ₂₂	m ₂₃
m ₃₁	m ₃₂	m ₃₃

Output Image

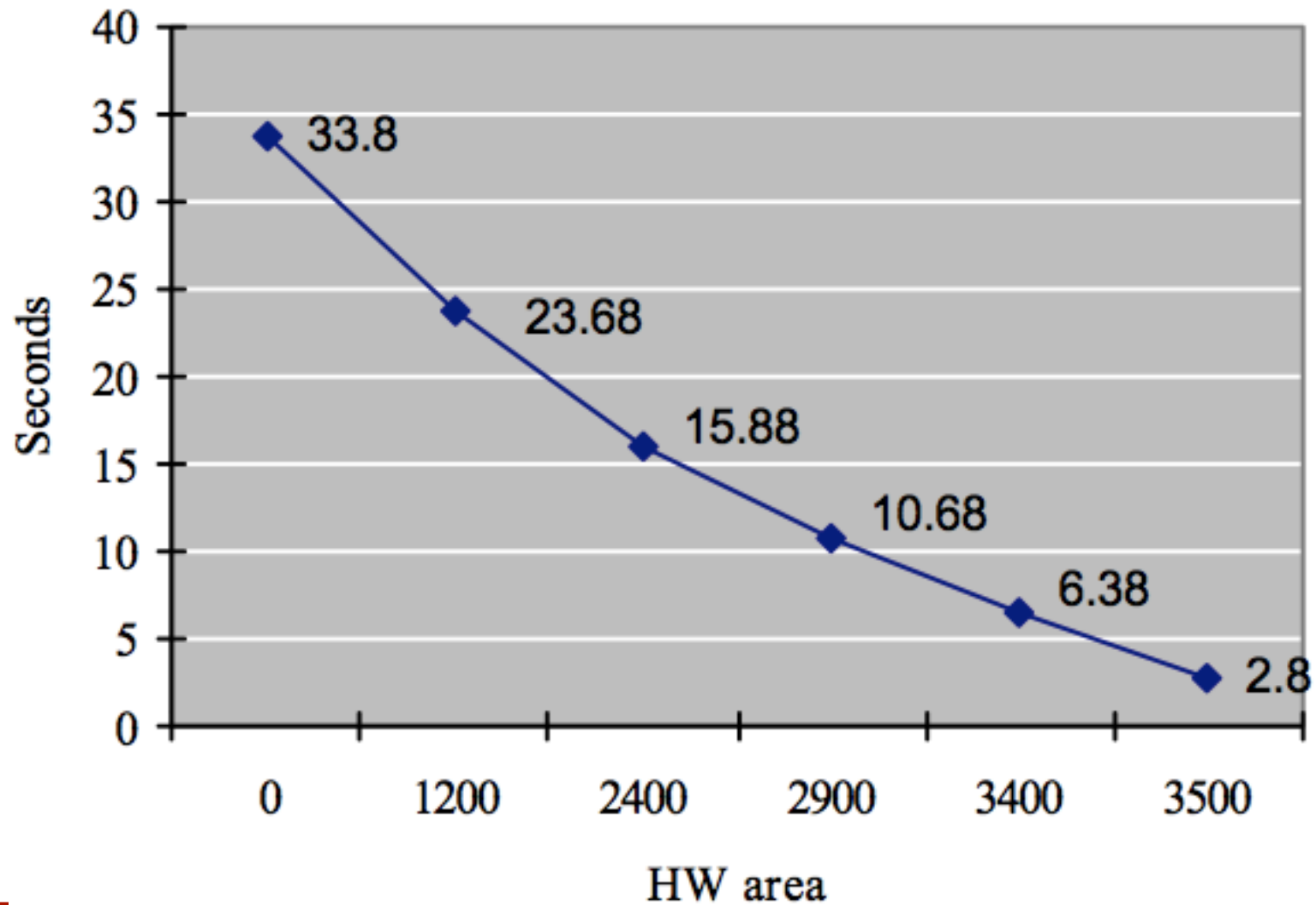
b ₁₁	b ₁₂	b ₁₃		
b ₂₁	b ₂₂	b ₂₃		
b ₃₁	b ₃₂	b ₃₃		

$$b_{22} = (a_{11} * m_{11}) + (a_{12} * m_{12}) + (a_{13} * m_{13}) + (a_{21} * m_{21}) + (a_{22} * m_{22}) + (a_{23} * m_{23}) + (a_{31} * m_{31}) + (a_{32} * m_{32}) + (a_{33} * m_{33})$$

SOBEL Edge Detection

```
main() {  
    unsigned char image_in[ROWS][COLS];  
    unsigned char image_out[ROWS][COLS];  
    int r, c; /* row and column array counters */  
    int pixel; /* temporary value of pixel */  
        /*filter the image and store result in output array */  
    for (r=1; r<ROWS-1; r++)  
        for (c=1; c<COLS-1; c++) { /* Apply Sobel operator. */  
            pixel = image_in[r-1][c+1]-image_in[r-1][c-1]  
                + 2*image_in[r][c+1] - 2*image_in[r][c-1]  
                + image_in[r+1][c+1] - image_in[r+1][c-1];  
            /* Normalize and take absolute value */  
            pixel = abs(pixel/4);  
            /* Check magnitude */  
            if (pixel > Threshold)  
                pixel= 255; /*EDGE_VALUE;*/  
            /* Store in output array */  
            image_out[r][c] = (unsigned char) pixel;  
        }  
    }
```

SOBEL Edge Detection



Final issues

- Come by my office hours (right after class)
- Any questions or concerns?