Design Process

• Describe system in programs

• Data subsystem
  – List data operations
  – Map operations to functional blocks
  – Add interconnect for data transport
  – Input control signals and output conditions

• Control Subsystem
  – Derive the sequence according to the hardware program
  – Create the sequential machine
  – Input conditions and output control signals
Example: Multiplication

Arithmetic

\[ Z = X \times Y \]

- \( M = 0 \)
- For \( i = n-1 \) to 0
  - If \( Y_i = 1 \), \( M = M + X \times 2^i \)
- \( Z = M \)

Input \( X, Y \)

Output \( Z \)

Variable \( M, i \)

- \( M = 0 \)
- For \( i = n-1 \) to 0
  - If \( Y_{n-1} = 1 \), \( M = M + X \)
  - Shift \( Y \) left by one bit
  - If \( i \neq 0 \), shift \( M \) left by one bit
- \( Z = M \)
Implementation: Example

Multiply(X, Y, Z, start, done)
{ Input X[15:0], Y[15:0] type bit-vector, start type Boolean;
  Local-Object A[15:0], B[15:0], M[31:0], i[4:0] type bit-vector;
  Output Z[31:0] type bit-vector, done type Boolean;
S0: If start’ goto S0 || done ← 1;
S1: A ← X || B ← Y || i ← 0 || M ← 0 || done ← 0;
S2: If B_{15} = 0 goto S4 || i ← i+1;
S3: M ← M+A;
S4: if i >= 16, goto S6
S5: M ← Shift(M,L,1) || B ← Shift(B,L,1) || goto S2;
S6: Z: ← M || done ← 1 || goto S0;
}
Step 0: Syntax

Multiply(X, Y, Z, start, done)
{ Input X[15:0], Y[15:0] type bit-vector, start type boolean;
  Local-Object A[15:0], B[15:0], M[31:0], i[4:0] type bit-vector;
  Output Z[31:0] type bit-vector, done type boolean;
S0: If start’ goto S0 || done ← 1;
S1: A ← X || B ← Y || i ← 0 || M ← 0 || done ← 0;
S2: If B_{15} = 0 goto S4 || i ← i+1;
S3: M ← M+A;
S4: if i≥ 16, goto S6
S5: M ← Shift(M,L,1) || B ← Shift(B,L,1) || goto S2;
S6: Z: ← M || done ← 1 || goto S0;
}
Multiply(X, Y, Z, start, done)
{ Input:  X[15:0], Y[15:0] type bit-vector,  start type boolean;
    Local-Object : A[15:0], B[15:0], M[31:0], i[4:0] type bit-vector;
    Output Z[31:0] type bit-vector, done type boolean;
S0: If start’ goto S0 || done  1;
S1: A  X || B  Y || i  0 || M  0 || done  0;
S2: If B_{15} = 0 goto S4 || i  i+1;
S3: M  M+A;
S4: if i>= 16, goto S6
S5: M  Shift(M,L,1) ||
    B  Shift(B,L,1) || goto S2;
S6: Z:  M || done  1|| goto S0;
}
Step 2: Identify Condition Bits to Control Subsystem

Multiply(X, Y, Z, start, done)
{ Input: X[15:0], Y[15:0] type bit-vector, start type boolean;
Local-Object : A[15:0], B[15:0], M[31:0], i[4:0] type bit-vector;
Output Z[31:0] type bit-vector, done type boolean;
S0: If start’ goto S0 || done ← 1;
S1: A ← X || B ← Y || i ← 0 || M ← 0 || done ← 0;
S2: If B_{15} = 0 goto S4 || i ← i+1;
S3: M ← M + A;
S4: if i >= 16, goto S6
S5: M ← Shift(M, L, 1) || B ← Shift(B, L, 1) || goto S2;
S6: Z: ← M || done ← 1 || goto S0;
}
**Step 3: Identify Data Subsystem Operations**

Multiply($X$, $Y$, $Z$, start, done)

\{
    \text{Input: } X[15:0], Y[15:0] \text{ type bit-vector, start type boolean;}
    \text{Local-Object: } A[15:0], B[15:0], M[31:0], i[4:0] \text{ type bit-vector;}
    \text{Output: } Z[31:0] \text{ type bit-vector, done type boolean;}
\}

- S0: If start’ goto S0 || done $\leftarrow$ 1;
- S1: $A \leftarrow X || B \leftarrow Y || i \leftarrow 0 || M \leftarrow 0 || done \leftarrow 0$;
- S2: If $B_{15} = 0$ goto S4 || $i \leftarrow i+1$;
- S3: $M \leftarrow M+A$;
- S4: if $i \geq 16$, goto S6
- S5: $M \leftarrow \text{Shift}(M,L,1) || B \leftarrow \text{Shift}(B,L,1) ||$ goto S2;
- S6: $Z: M || done \leftarrow 1 ||$ goto S0;
\}
Step 4: Map Data Operations to Implementable functions

Multiply(X, Y, Z, start, done)
{
    Input: X[15:0], Y[15:0] type bit-vector, start type boolean;
    Local-Object: A[15:0], B[15:0], M[31:0], i[4:0] type bit-vector;
    Output Z[31:0] type bit-vector, done type boolean;

    S0: If start’ goto S0 || done  1;
    S1: A  X || B  Y || i  0 || M  0 || done  0;
    S2: If B_15 = 0 goto S4 || i  i+1;
    S3: M  M+A;
    S4: if i >= 16, goto S6
    S5: M  Shift(M,L,1) || B  Shift(B,L,1) || goto S2;
    S6: Z:  M || done  1 || goto S0;
}
Step 5: Implement the Data Subsystem from Standard Modules

Registers: If C then R $\leftarrow$ D

operation

- $A \leftarrow \text{Load (X)}$
- $B \leftarrow \text{Load (Y)}$
- $M \leftarrow \text{Clear(M)}$
- $i \leftarrow \text{Clear}(i)$
- $i \leftarrow \text{INC}(i)$
- $M \leftarrow \text{Add}(M,A)$
- $M \leftarrow \text{SHL}(M)$
- $B \leftarrow \text{SHL}(B)$
Storage Component: Registers with control signals

Registers: If C then R ← D

operation
A ← Load (X)
B ← Load (Y)
M ← Clear(M)
i ← Clear(i)
i ← INC(i)
M ← Add(M, A)
M ← SHL(M)
B ← SHL(B)
Data Subsystem

operation
A ← Load (X)
B ← Load (Y)
M ← Clear(M)
i ← Clear(i)
i ← INC(i)
M ← Add(M, A)
M ← SHL(M)
B ← SHL(B)
Function Modules: Adder, Shifter

- **A** ← Load (X)
- **B** ← Load (Y)
- **M** ← Clear(M)
- **i** ← Clear(i)
- **i** ← INC(i)
- **M** ← Add(M,A)
- **M** ← SHL(M)
- **B** ← SHL(B)

Registers B and M have multiple sources.
Function Modules: Adder, Shifter, Counter

operation
A ← Load (X)
B ← Load (Y)
M ← Clear(M)
i ← Clear(i)
i ← INC(i)
M ← Add(M, A)
M ← SHL(M)
B ← SHL(B)
Step 6: Map Control Signals to Operations

operation
A ← Load (X)
B ← Load (Y)
M ← Clear(M)
i ← Clear(i)
i ← INC(i)
M ← Add(M,A)
M ← SHL(M)
B ← SHL(B)

control
C₀ = 1
C₂ = 0 and C₅ = 1
C₃ = 1
C₆ = 1
C₇ = 1
C₁ = 0 and C₄ = 1
C₁ = 1 and C₄ = 1
C₂ = 1 and C₅ = 1

Diagram:

- Register A
- Register M
- Register B
- Counter i
- Selector
Step 7: Identify Control Path Components

Multiply(X, Y, Z, start, done)
{ Input: X[15:0], Y[15:0] type bit-vector,
   start type boolean;
 Local-Object : A[15:0], B[15:0], M[31:0],
   i[4:0] type bit-vector;
 Output: Z[31:0] type bit-vector,
   done type boolean;
 S0: If start’ goto S0 || done ← 1;
 S1: A ← X || B ← Y || i ← 0 || M ← 0 || done ← 0;
 S2: If B_{15} = 0 goto S4 || i ← i+1;
 S3: M ← M+A;
 S4: if i >= 16, goto S6
 S5: M ← Shift(M, L, 1) || B ← Shift(B, L, 1) || goto S2;
 S6: Z ← M || done ← 1 || goto S0;
}

Z = XY

The diagram illustrates the data flow and control logic for the multiplication operation, showing how the input values (X, Y, Z) and control signals (start, done) interact within the data and control subsystems.

The control unit processes the control signals to manage the flow of data through the data subsystem, ensuring the correct operations are performed at each step.
Data Subsystem

X $\rightarrow$ 16
Y $\rightarrow$ 16

B[15], i[4] $\rightarrow$ C_{0:7}

start $\rightarrow$ Control Subsystem

32 $\rightarrow$ Z

Control Subsystem

done
operation | control
---|---
A ← Load (X) | C₀ = 1
B ← Load (Y) | C₂ = 0 and C₅ = 1
M ← Clear(M) | C₃ = 1
i ← Clear(i) | C₆ = 1
i ← INC(i) | C₇ = 1
M ← Add(M,A) | C₁ = 0 and C₄ = 1
M ← SHL(M) | C₁ = 1 and C₄ = 1
B ← SHL(B) | C₂ = 1 and C₅ = 1

Multiply(X, Y, Z, start, done) {
S0: If start′ goto S0 || done ← 1;
S1: A ← X || B ← Y || i ← 0 || M ← 0 || done ← 0;
S2: If B₁₅ = 0 goto S4 || i ← i+1;
S3: M ← M+A;
S4: if i >= 16, goto S6
S5: M ← Shift(M,L,1) || B ← Shift(B,L,1) || goto S2;
S6: Z ← M || done ← 1 || goto S0;
}

<table>
<thead>
<tr>
<th></th>
<th>C₀ Load A</th>
<th>C₁ Feed M</th>
<th>C₂ Feed B</th>
<th>C₃ Clr M</th>
<th>C₄ Load M</th>
<th>C₅ Load B</th>
<th>C₆ Clr i</th>
<th>C₇ Inc i</th>
<th>done</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S₁</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₂</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₄</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₅</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₆</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Multiply(X, Y, Z, start, done) {
S0: If start’ goto S0 || done ← 1;
S1: A ← X || B ← Y || i ← 0 || M ← 0 || done ← 0;
S2: If B_{15} = 0 goto S4 || i ← i+1;
S3: M ← M+A;
S4: if i ≥ 16, goto S6
S5: M ← Shift(M, L, 1) || B ← Shift(B, L, 1) || goto S2;
S6: Z: ← M || done ← 1|| goto S0;
}

### Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ← Load (X)</td>
<td>C₀ = 1</td>
</tr>
<tr>
<td>B ← Load (Y)</td>
<td>C₂ = 0 and C₅ = 1</td>
</tr>
<tr>
<td>M ← Clear(M)</td>
<td>C₃ = 1</td>
</tr>
<tr>
<td>i ← Clear(i)</td>
<td>C₆ = 1</td>
</tr>
<tr>
<td>i ← INC(i)</td>
<td>C₇ = 1</td>
</tr>
<tr>
<td>M ← Add(M, A)</td>
<td>C₁ = 0 and C₄ = 1</td>
</tr>
<tr>
<td>M ← SHL(M)</td>
<td>C₁ = 1 and C₄ = 1</td>
</tr>
<tr>
<td>B ← SHL(B)</td>
<td>C₂ = 1 and C₅ = 1</td>
</tr>
</tbody>
</table>

### Table

<table>
<thead>
<tr>
<th></th>
<th>C₀ (Load A)</th>
<th>C₁ (Feed M)</th>
<th>C₂ (Feed B)</th>
<th>C₃ (Clr M)</th>
<th>C₄ (Load M)</th>
<th>C₅ (Load B)</th>
<th>C₆ (Clr i)</th>
<th>C₇ (Inc i)</th>
<th>done</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S₁</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₂</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₄</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₅</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S₆</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Design of the Control Subsystem

Multiply(X, Y, Z, start, done)
{
S0: If start’ goto S0 || done <- 1;
S1: A <- X || B <- Y || i <- 0 || M <- 0 ||
done <- 0;
S2: If B_{15} = 0 goto S4 || i <- i+1;
S3: M <- M+A;
S4: if i >= 16, goto S6
S5: M <- Shift(M,L,1) ||
B <- Shift(B,L,1) || goto S2;
S6: Z: <- M || done <- 1|| goto S0
}
Control Subsystem

Multiply(X, Y, Z, start, done)  
{  
  S0: If start’ goto S0 || done<-1;  
  S1: A<-X || B<-Y || i<-0 || M<-0 ||  
     done<-0;  
  S2: If B_{15} = 0 goto S4 || i<-i+1;  
  S3: M<-M+A;  
  S4: if i>= 16, goto S6  
  S5: M<-Shift(M,L,1) ||  
     B<-Shift(B,L,1) || goto S2;  
  S6: Z<-M || done<-1|| goto S0  
}
## State Assignment

<table>
<thead>
<tr>
<th>Binary</th>
<th>$b_2b_1b_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>000</td>
</tr>
<tr>
<td>S1</td>
<td>001</td>
</tr>
<tr>
<td>S2</td>
<td>010</td>
</tr>
<tr>
<td>S3</td>
<td>011</td>
</tr>
<tr>
<td>S4</td>
<td>100</td>
</tr>
<tr>
<td>S5</td>
<td>101</td>
</tr>
<tr>
<td>S6</td>
<td>110</td>
</tr>
<tr>
<td>S7</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gray</th>
<th>$b_2b_1b_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>000</td>
</tr>
<tr>
<td>S1</td>
<td>001</td>
</tr>
<tr>
<td>S2</td>
<td>011</td>
</tr>
<tr>
<td>S3</td>
<td>010</td>
</tr>
<tr>
<td>S4</td>
<td>110</td>
</tr>
<tr>
<td>S5</td>
<td>111</td>
</tr>
<tr>
<td>S6</td>
<td>101</td>
</tr>
<tr>
<td>S7</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>One Hot</th>
<th>$b_7b_6b_5b_4b_3b_2b_1b_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>S1</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>S2</td>
<td>0 0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>S3</td>
<td>0 0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>S4</td>
<td>0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>S5</td>
<td>0 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>S6</td>
<td>0 1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>S7</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

One Hot Encoding: n bits for n states. Bit $i=1$ for state $i$. 
Control Subsystem: One-Hot State Machine Design

Input: State Diagram
1. Use a flip flop to replace each state.
   Set the flip flop which corresponds to the initial state and reset the rest flip flops.
2. Use an OR gate to collect all inward edges.
3. Use a Demux to distribute the outward edges.
One-Hot State Machine

Start

S0

S1

S2

S3

S4

S5

S6

start

start'

B[15]

B[15']

i[4]

i[4']

S0

S1

S2

S3

S4

S5

S6

start'

start

B15

B15'

i[4]

i[4']
Control Subsystem: One-Hot State Machine Design

Input: State Diagram
1. Use a flip flop to replace each state. Set the flip flop which corresponds to the initial state and reset the rest flip flops.
2. Use an OR gate to collect all inward edges.
3. Use a Demux to distribute the outward edges.
Implementation: Example

Given a hardware program, implement data path and control subsystems

{ Input X[7:0], Y[7:0] type bit-vector, start type boolean;
  Local-Object A[7:0], B[7:0] type bit-vector;
  Output Z[7:0] type bit-vector, done type boolean;
  Wait: If start’ goto Wait || done←1;
    S1: A ← X || B ← Y || done ← 0;
    S2: If B >= 0 goto S4;
    S3: B ← -B;
    S4: If A >= B goto S6;
    S5: A ← A + 1 || B ← B-1 || goto S4;
    S6: Z ← 4 * A || done ← 1 || goto Wait;
}
Step 1: Identify Input and Output of data and control subsystems

Some_function
{ Input X[7:0], Y[7:0] type bit-vector,
  start type boolean;
Local-Object A[7:0], B[7:0] type bit-vector;
Output Z[7:0] type bit-vector,
  done type boolean;
Wait: If start' goto Wait || done←1;
S1: A ←X || B ←Y|| done ←0;
S2: If B >= 0 goto S4;
S3: B ←-B;
S4: If A >= B goto S6;
S5: A ←A + 1 || B ←B-1 || goto S4;
S6: Z ←4 * A || done ←1 || goto Wait;
}
Step 2: Identify Data Subsystem Operations

Some_function
{
    Input X[7:0], Y[7:0] type bit-vector,
    start type boolean;
    Local-Object A[7:0], B[7:0] type bit-vector;
    Output Z[7:0] type bit-vector,
    done type boolean;
    Wait: If start’ goto Wait || done<-1;
    S1: A <- X || B <- Y|| done <- 0;
    S2: If B >= 0 goto S4;
    S3: B <- -B;
    S4: If A >= B goto S6;
    S5: A <- A + 1 || B <- B-1 || goto S4;
    S6: Z <- 4 * A || done <- 1 || goto Wait;
}

\[
Z = \begin{cases} 
4 \ \text{Ceiling}(\frac{(X + |Y|)}{2}) & \text{if } X < |Y| \\
4X & \text{otherwise}
\end{cases}
\]
Step 2: Identify Data Subsystem Operations

Some_function
{ Input X[7:0], Y[7:0] type bit-vector,
  start type boolean;
  Local-Object A[7:0], B[7:0] type bit-vector;
  Output Z[7:0] type bit-vector,
  done type boolean;
Wait: If start’ goto Wait || done <- 1;
  S1: A ← X || B ← Y || done <= 0;
  S2: If B >= 0 goto S4;
  S3: B ← -B;
  S4: If A >= B goto S6;
  S5: A ← A + 1 || B ← B-1 || goto S4;
  S6: Z ← 4 * A || done ← 1 || goto Wait;
}
Step 2: Map Data Operations to Implementable functions

{Input X[7:0], Y[7:0] type bit-vector,
start type boolean;
Local-Object A[7:0], B[7:0] type bit-vector;
Output Z[7:0] type bit-vector,
done type boolean;
Wait: If start’ goto Wait || done⇐ 1;
S1: A ⇐ X || B ⇐ Y || done ⇐ 0;
S2: If B >= 0 goto S4;
S3: B ⇐ -B;
S4: If A >= B goto S6;
S5: A ⇐ A + 1 || B ⇐ B-1 || goto S4;
S6: Z ⇐ 4 * A || done ⇐ 1 || goto Wait;
}

<table>
<thead>
<tr>
<th>operation</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⇐ X</td>
<td>A ⇐ Load (X)</td>
</tr>
<tr>
<td>B ⇐ Y</td>
<td>B ⇐ Load (Y)</td>
</tr>
<tr>
<td>B ⇐ -B</td>
<td>B ⇐ CS (B)</td>
</tr>
<tr>
<td>A &gt;= B</td>
<td>Comp (A, B)</td>
</tr>
<tr>
<td>A ⇐ A + 1</td>
<td>A ⇐ INC (A)</td>
</tr>
<tr>
<td>B ⇐ B - 1</td>
<td>B ⇐ DEC (B)</td>
</tr>
<tr>
<td>Z ⇐ 4A</td>
<td>Z ⇐ SHL(A)</td>
</tr>
</tbody>
</table>
Step 3: Tag each Data Operations with a Control Signal

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ← X</td>
<td>Load (X)</td>
</tr>
<tr>
<td>B ← Y</td>
<td>Load (Y)</td>
</tr>
<tr>
<td>B ← -B</td>
<td>CS (B)</td>
</tr>
<tr>
<td>A ≥ B</td>
<td>Comp (A, B)</td>
</tr>
<tr>
<td>A ← A + 1</td>
<td>INC (A)</td>
</tr>
<tr>
<td>B ← B - 1</td>
<td>DEC (B)</td>
</tr>
<tr>
<td>Z ← 4A</td>
<td>SHL(A)</td>
</tr>
</tbody>
</table>

Data Subsystem

Control Subsystem

X → 8
Y → 8

start → Data Subsystem

? → Control Subsystem

8 vs. 8

Z → done
Step 4: Identify Condition Bits to Control Subsystem

{Input X[7:0], Y[7:0] type bit-vector, start type boolean;  
Local-Object A[7:0], B[7:0] type bit-vector;  
Output Z[7:0] type bit-vector, done type boolean;  
Wait: If start’ goto Wait || done⇐1;  
S1: A ⇐ X || B ⇐ Y|| done ⇐ 0;  
S2: If B ≥ 0 goto S4;  
S3: B ⇐ -B;  
S4: If A ≥ B goto S6;  
S5: A ⇐ A + 1 || B ⇐ B-1 || goto S4;  
S6: Z ⇐ 4 * A || done⇐ 1 || goto Wait;  
}
Step 5: Implement the Data Subsystem from Standard Modules

operation
A ← Load (X)
B ← Load (Y)
B ← CS (B)
Comp (A, B)
A ← INC (A)
B ← DEC (B)
Z ← SHL(A)
Step 6: Map Control Signals to Operations
Step 7: Identify Control Path Components

S0: If start’, goto S0, else goto S1 || done ← 1
S1: A ← X || B ← Y || done ← 0 || goto S2
S2: If B’<7> goto S4, else goto S3
S3: B ← CS (B) || goto S4
S4: If k goto S6, else goto S5
S5: A ← INC (A) || B ← DEC (B) || goto S4
S6: Z ← A || goto S7
S7: Z ← SHL (z) || goto S8
S8: Z ← SHL (z) || done ← 1 || goto S0
One-Hot State Machine
S0: If start’, goto S0, else goto S1 || done<=1
S1: A ← X || B ← Y || done ← 0 || goto S2
S2: If B’<7> goto S4, else goto S3
S3: B ← CS (B) || goto S4
S4: If k goto S6, else goto S5
S5: A ← INC (A) || B ← DEC (B) || goto S4
S6: Z ← A || goto S7
S7: Z ← SHL (z) || goto S8
S8: Z ← SHL (z) || done<=1 || goto S0
Summary

• Hardware Allocation
  • Balance between cost and performance
• Resource Sharing and Binding
  • Map operations to hardware
• Interconnect Synthesis
  • Convey signal transports
• Operation Scheduling
  • Sequence the process
Remarks:
1. Implement the control subsystem with one-hot state machine design.
2. Try to reduce the latency of the whole system.