EDA322
Digital Design

Lecture 13:
System Design and Interfaces,

Ioannis Sourdis

Outline of Lecture 13

- System Design process
- Interfaces

System Design – a process

- Specification
  - Understand what you need to build
- Divide and conquer
  - Break it down into manageable pieces
- Define interfaces
  - Clearly specify every signal between pieces
  - Hide implementation
  - Choose representations
- Timing and sequencing
  - Overall timing – use a table
  - Timing of each interface – use a simple convention (e.g., valid – ready)
- Add parallelism as needed (pipeline or duplicate units)
- Timing and sequencing (of parallel structures)
- Design each module
- Code
- Verify

Iterate back to the top at any step as needed.

Specification

- Write the user's manual first

- Putting it on paper means that there are no misunderstandings about operation
  - In practice, this also serves to validate the specification with users/customers

- Spec includes
  - Inputs and outputs
  - Operating modes
  - Visible state
  - Discussion of "edge cases"

- Most of design is done writing English-language documents – with associated drawings. Coding comes later.
 - Don't start coding until your design is complete.

Divide and Conquer –common themes

- Task
  - Divide system into a network of tasks
  - One module per task
  - ***Model-view-controller***: tasks are:
    - The 'guts' (model)
    - Output modules that 'view' the model
    - Input modules that affect the model
- State
  - Divide system by state
  - Separate module for each set of state variables
- Interface
 - Module for each external interface

Divide and Conquer

- Example 1 – Pong
  - Model-view-controller
  - Model – ball and paddle position FSMs, score
  - View – VGA display and sound
  - Controller – inputs to control paddles

# Pong Decomposition

Pong

Buttons

RGBSync
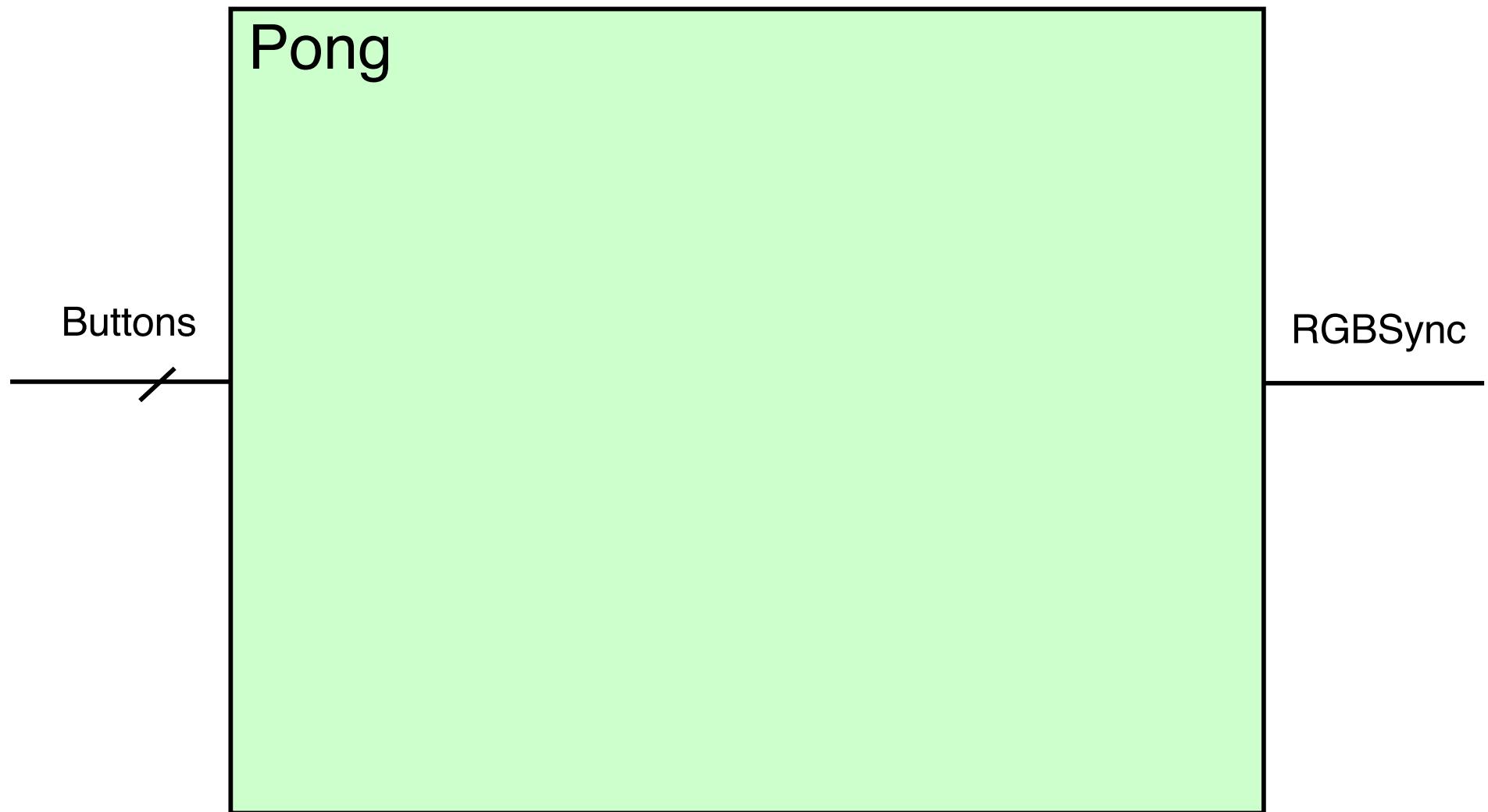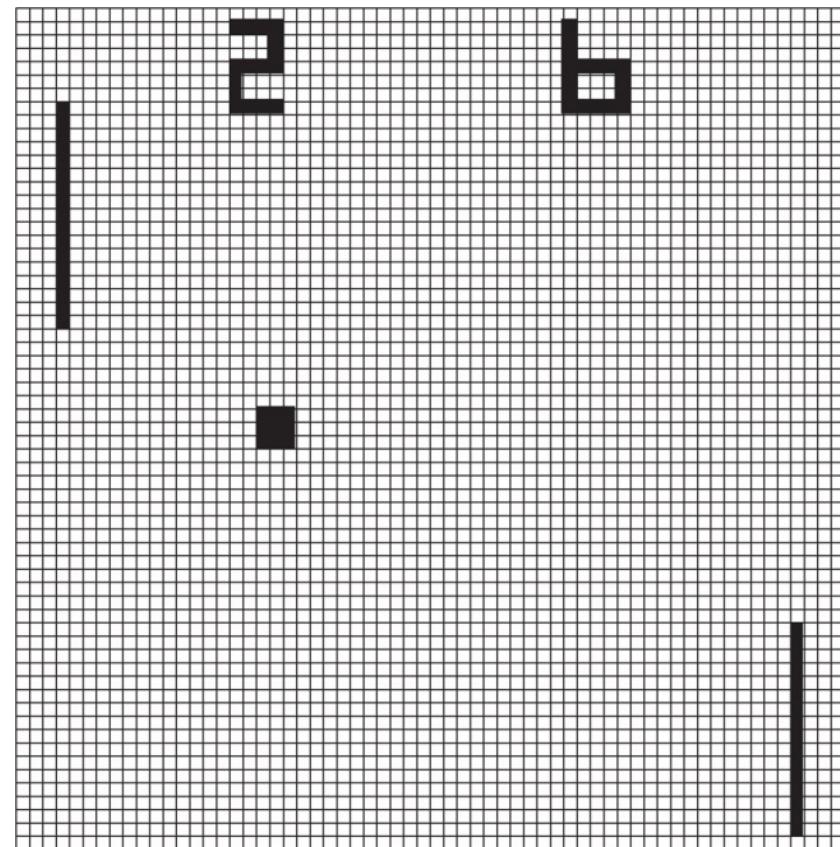
Pong

Pong Decomposition
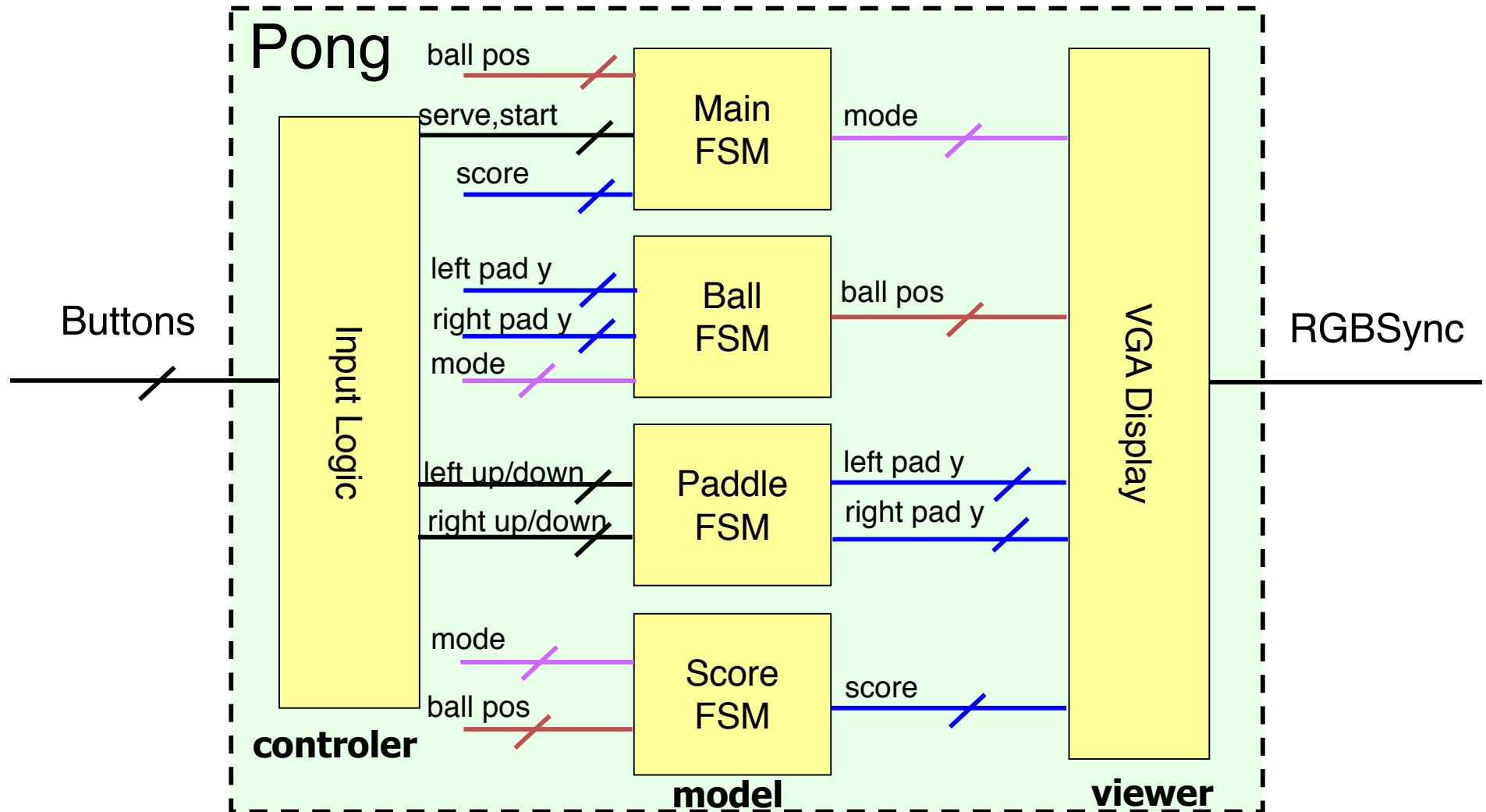
**Horizontally partitioned by task**

Model partitions also vertically by state

Each block is now small enough to design

- Example, Ball FSM
  - State: x, y, vx, vy
  - Serve: {x, y} = middle, vy = 0, vx = dir ;
  - Bounce off top/bottom: vy = -vy ;
  - Bounce off paddle: vx = -vx ; adjust vy ;
  - Otherwise: x = x+vx, y = y+vy ;

- Simple datapath FSM

Divide and Conquer

- Example 2 – DES Cracker
  - Task pipeline:
    - Generate keys
    - Sequence ciphertext
    - Decrypt plaintext
   - Check plaintext

DES Example
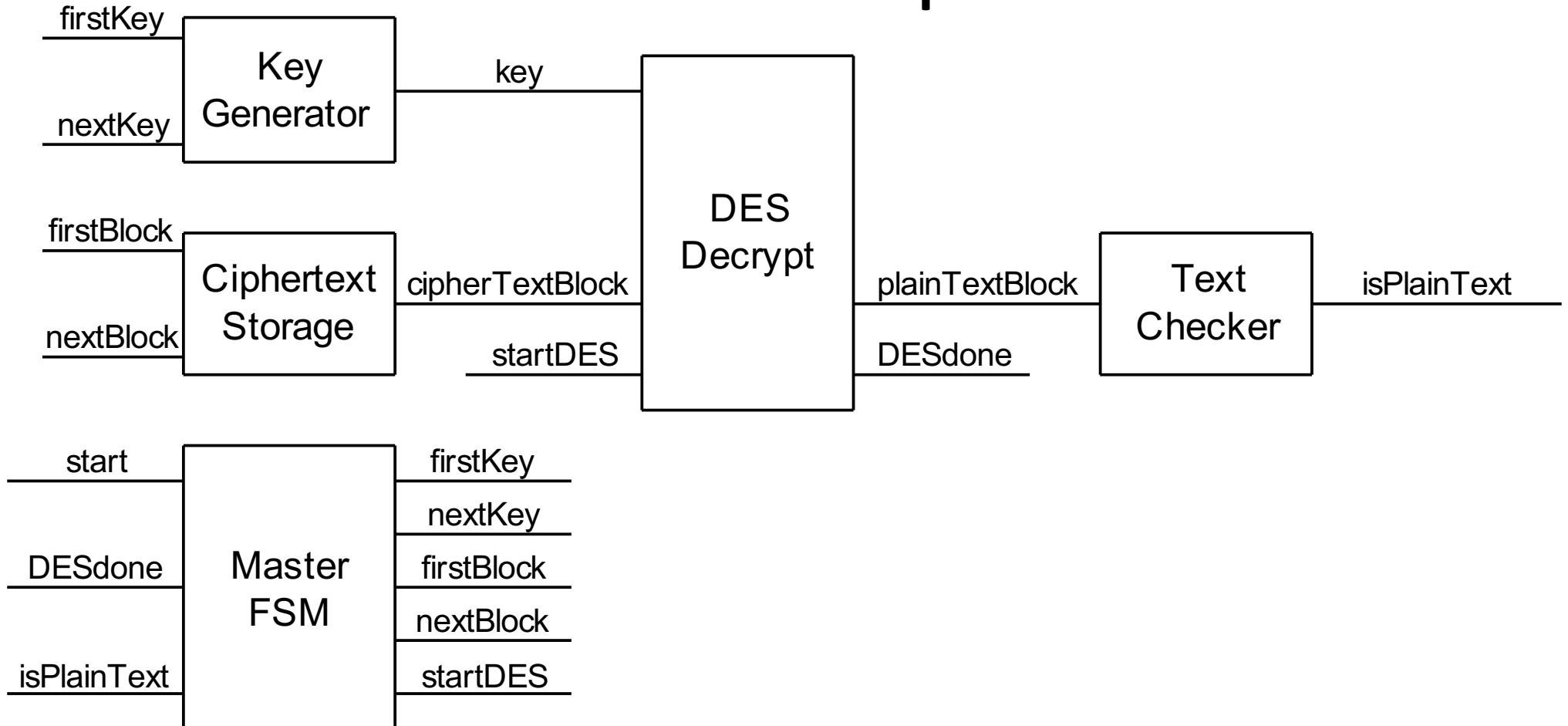
Master-slave partitioning

Divide and Conquer

- Example 3 – Music Synthesizer
  - State x task
    - Tone generator
      - Generate harmonics, generate addresses, lookup sine wave, weight for each harmonic
    - Envelope generator
      - Generate envelopes, multiply by samples
    - Combiner

Simple Music Synthesizer
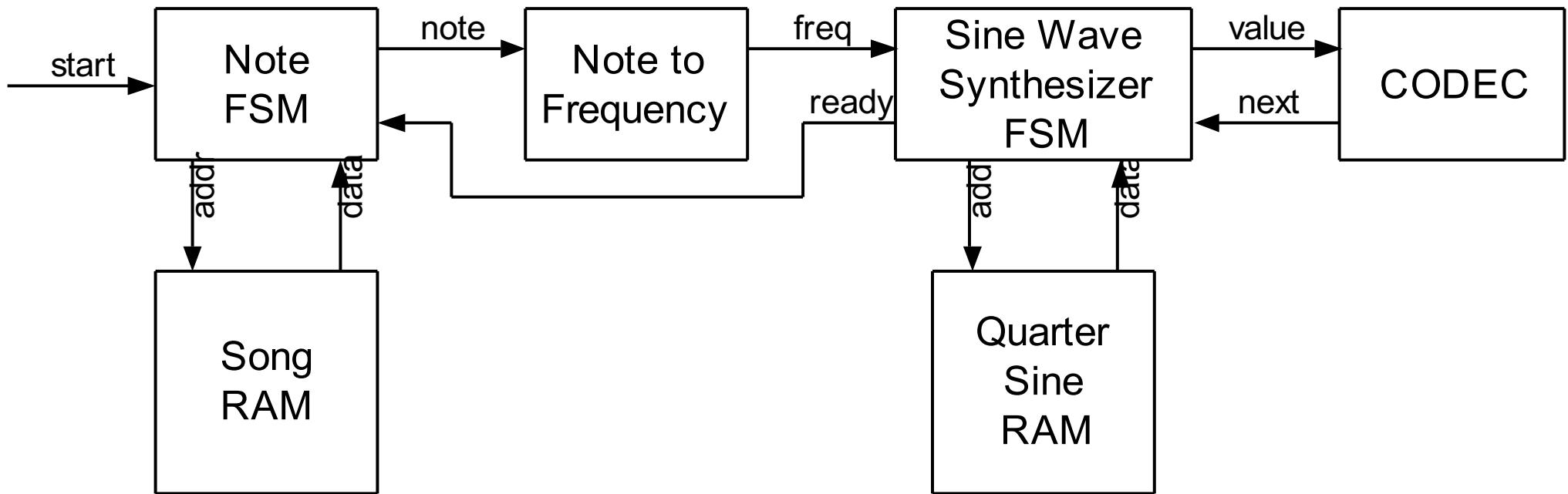
- Note FSM: determines the next note to play
- Note-to-Freq: converts the note into a frequency
- Sine-wave synthesizer FSM: synthesizes sine wave of the specified frequency

With Harmonics and Attack/Decay
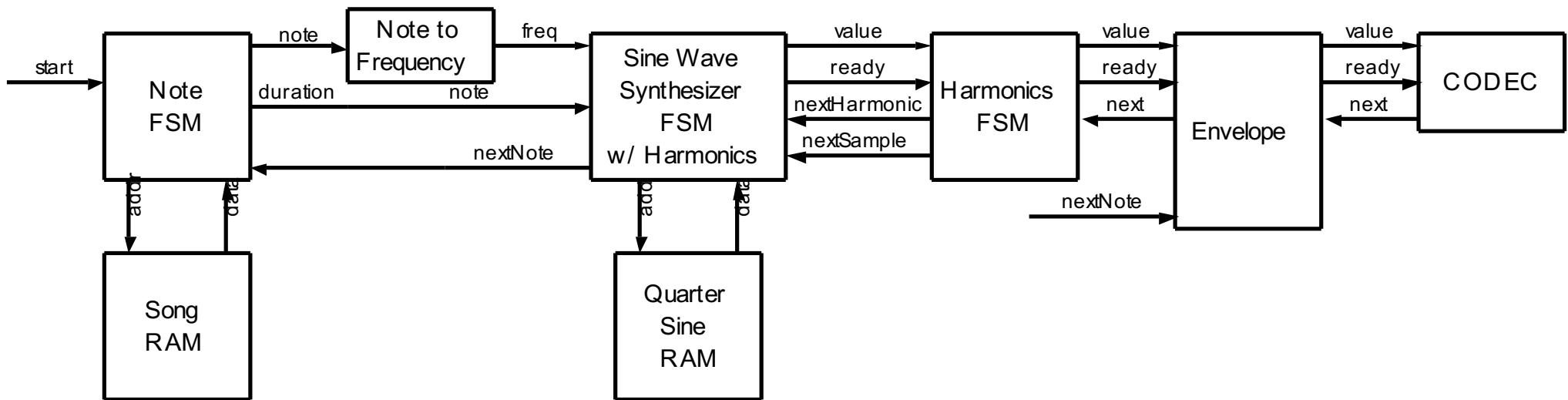
- 2 new FSMs to compute harmonics & an attenuation envelope

- Harmonic

- Harmonic FSM receives from synthesizer FSM series of S0.15 waveform values

- It combines these waves as input to the envelope FSM

- Envelope FSM attenuates the wave with an attack-decay envelope using the current time-step

- Time-counter resets when "NextNote"

Define Interfaces

- For standard modules, already defined for you
  - DES module (from OpenCores library)
  - AC97 CODEC

- For your own modules, interfaces must specify every signal
  - Each Data "Port":
    - Data signals – how wide, what representation, when valid
    - Flow control – specifies when data transfers take place
  - Other control and status
  - Example – Sine Wave Generator
    - Next (in) – goes high one clock each data sample
    - Freq (in) – 16-bits – u0.16 specifies an interval between samples in the sine table. A value of 1 specifies an interval of pi.
    - Value (out) – 16-bits s0.15 format, on sample pulse
    - NextNote (out) – goes high when current note has been held for 100ms

Example decision

- Suppose we need 15 sine-wave generators
  - 3 notes x 5 harmonics each
- Do we share a single quarter sine table or use 15 tables?

- In favor of sharing
  - We have time
    - Sample rate is 48KHz, clock is 100MHz.
    - 2,083 cycles per sample
  - It will take less chip area
- Opposed to sharing
  - Dedicated BRAMs are simpler
  - We have lots of BRAMs.

Which would you do?

System Design – a process (reminder)

- Specification
  - Understand what you need to build
- Divide and conquer
  - Break it down into manageable pieces
- Define interfaces
  - Clearly specify every signal between pieces
  - Hide implementation
  - Choose representations

- Timing and sequencing

- Add parallelism as needed (pipeline or duplicate units)

- Timing and sequencing (of parallel structures)

- Design each module

- Code

- Verify

Iterate back to the top at any step as needed.

Timing and Sequencing

- Work out exactly when and in what order things happen
- Account for pipeline delays
- Account for multi-cycle operations
- Draw a timing diagram (or a table)
- Example: DES Cracker

Example, DES Cracker Timing

Cycle	KeyGen	CT Seq	DES	Check
0	firstKey	firstBlock		
1	Key0	CT Block 0	Round 1	
2			Round 2	
…			…	
16		nextBlock	Round 16	
17		CT Block 1	Round 1	PT Block 0
18			Round 2	
…			…	
32		nextBlock	Round 16	
33		CT Block 2	Round 1	PT Block 1
…			…	
112		nextBlock	Round 16	
113		CT Block 7	Round 1	PT Block 6
…			…	
128	nextKey	firstBlock	Round 16	
129	Key1	CT Block 0		PT Block 7

DES Example
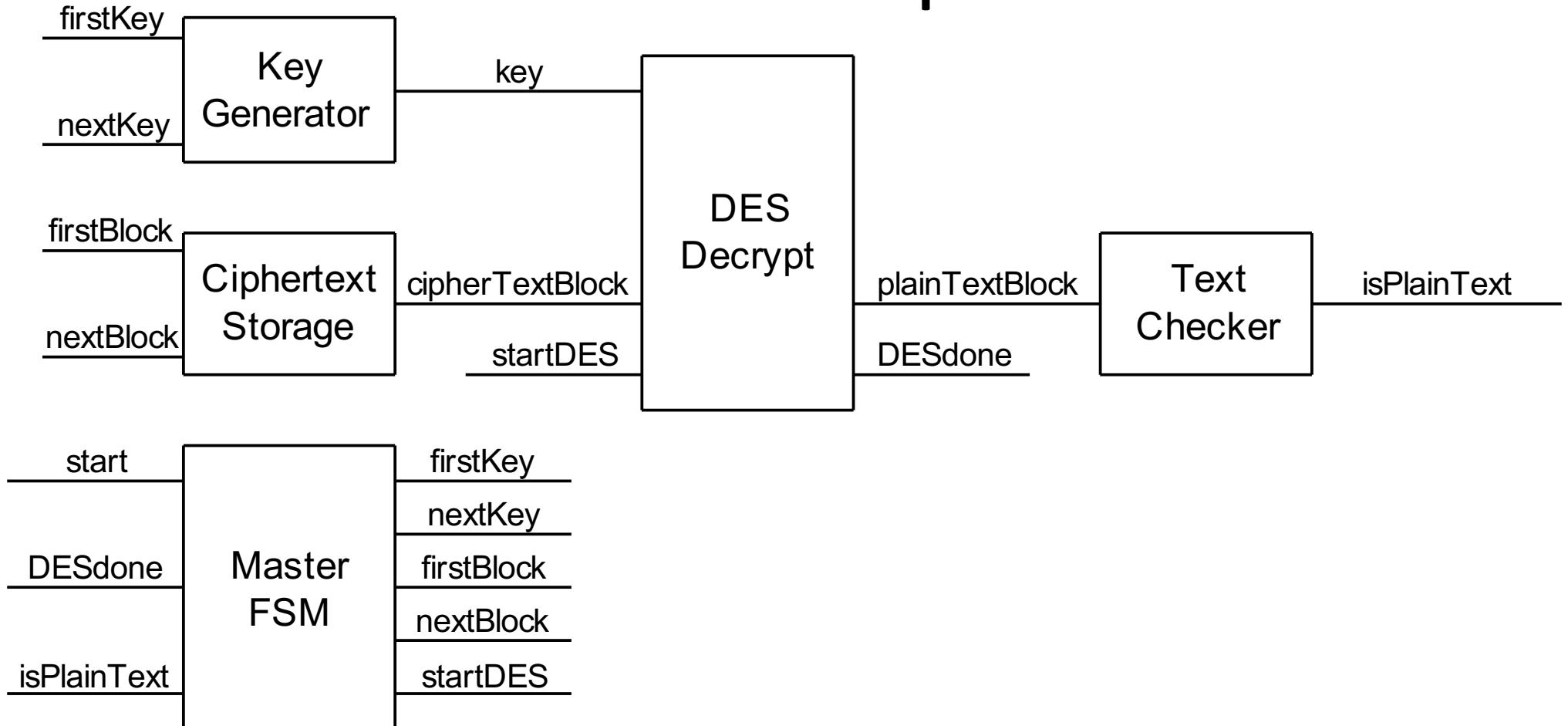
Example, DES Cracker Timing – if a PT block fails, go on to next key

Cycle	KeyGen	CT Seq	DES	Check
0	firstKey	firstBlock		
1	Key0	CT Block 0	Round 1	
2			Round 2	
…			…	
16		nextBlock	Round 16	
17		CT Block 1	Round 1	PT Block 0
18	nextKey	firstBlock	---	
19	Key1	CT Block 0	Round 1	
			Round 2	
…			…	
34		nextBlock	Round 16	
35		CT Block 1	Round 1	PT Block 0
…			…	

Example timing –
Music Synthesizer with Harmonics

Cycle	NextNote	NextSample	NextHarmonic	Ready	Comment
0	1	1			Start – look up note, convert to freq
1		1			Freq valid this cycle, read value
2		1		1	Value of fundamental
3			1		Read 2nd harmonic
4			1	1	Value of 2nd harmonic (2x freq)
5			1		Read 3rd harmonic
6			1	1	Value of 3rd harmonic (3x freq)
…					Idle until next 48KHz request
2084		1			Read fundamental for next sample
2085		1		1	Value of fundamental
2086			1		Read 2nd harmonic
2087			1	1	Value of 2nd harmonic (2x freq)
2088			1		Read 3rd harmonic
2089			1	1	Value of 3rd harmonic (3x freq)
…					Repeat above 4800 times per note
X+0	1	1			Read next note
X+2		1			Freq valid, read value
X+3		1		1	Value of fundamental

With Harmonics and Attack/Decay
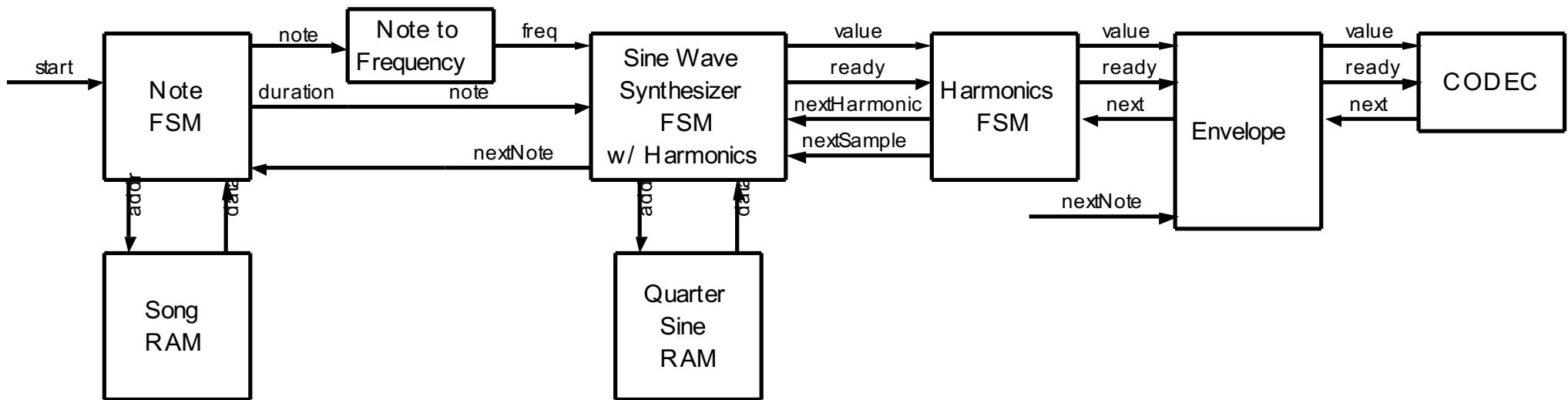
System Design – a process (reminder)

- Specification
  - Understand what you need to build
- Divide and conquer
  - Break it down into manageable pieces
- Define interfaces
  - Clearly specify every signal between pieces
  - Hide implementation
  - Choose representations
- Timing and sequencing
- Add parallelism as needed (pipeline or duplicate units)
- Timing and sequencing (of parallel structures)
- **Design each module**
- **Code**
- **Verify**

Iterate back to the top at any step as needed.

Basic principle

Keep it simple (KIS)

- Add complexity only when your design absolutely needs it

A corollary:

- If its not broken, don't fix it
- Don't optimize something unless there is something wrong with the simple design

System Design – a process (reminder)

- Specification
  - Understand what you need to build
- Divide and conquer
  - Break it down into manageable pieces
- Define interfaces
  - Clearly specify every signal between pieces
  - Hide implementation
  - Choose representations
- Timing and sequencing
- Add parallelism as needed (pipeline or duplicate units)
- Timing and sequencing (of parallel structures)

- Design each module
- Code
- Verify

Iterate back to the top at any step as needed.

Some comments on Coding

- Don't start coding until your design is done.
- Don't even think about coding until your design is done

- Code a separate module for every block in your block diagrams

- Unit test each module before moving on to the next module

- Follow good VHDL coding practice
  - Don't forget its hardware

Verification

- ## Basic principle
  - ### If you didn't test it, it doesn't work
    - All modules
    - All states
    - All transitions between states
    - All "edge" conditions

- ## Accelerate tests
  - ### Initialize to just before the state you're testing
  - ### Shorten counters (for testing, don't forget to lengthen for real operation)

Debugging

- Thinking your design through ahead of time will avoid most bugs
  - Work out timing
  - Keep it simple

- Be a detective to track down the few bugs that slip through
  - Start with known working logic
  - Follow signals to the point where something *first* goes wrong
  - Run simplest possible test case
  - Unit test modules

- Make sure you don't have compilation or simulation warnings
  - Check that your logic meets timing

- Do not just randomly change Verilog code - stop and think

System Design – Overview

- Specification
  - Understand what you need to build
- Divide and conquer
  - Break it down into manageable pieces
- Define interfaces
  - Clearly specify every signal between pieces
  - Hide implementation
  - Choose representations
- Timing and sequencing
- Add parallelism as needed (pipeline or duplicate units)
- Timing and sequencing (of parallel structures)
- Design each module
- Code
- Verify
  - Debug

Iterate back to the top at any step as needed.

Interface Timing

Interface Timing

- How do you pass data from one module to another?
  - Open loop
 - Flow control
  - Serialized
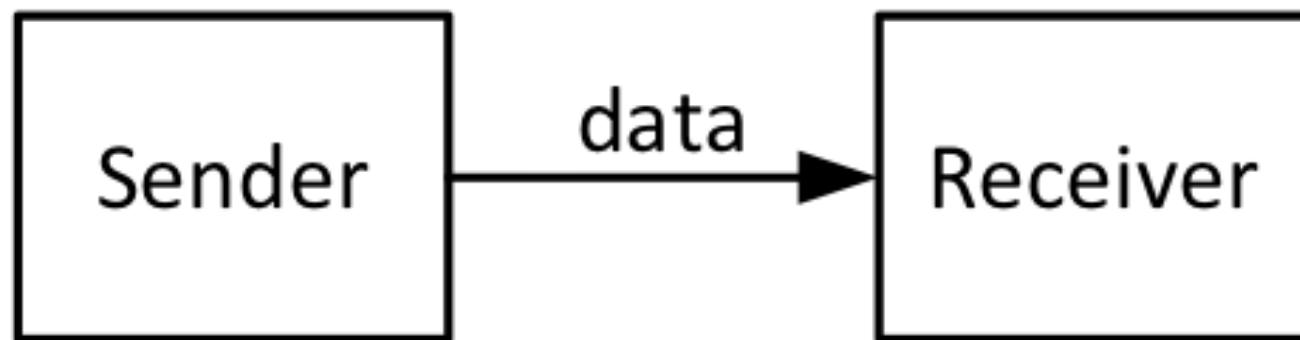
Always Valid Timing
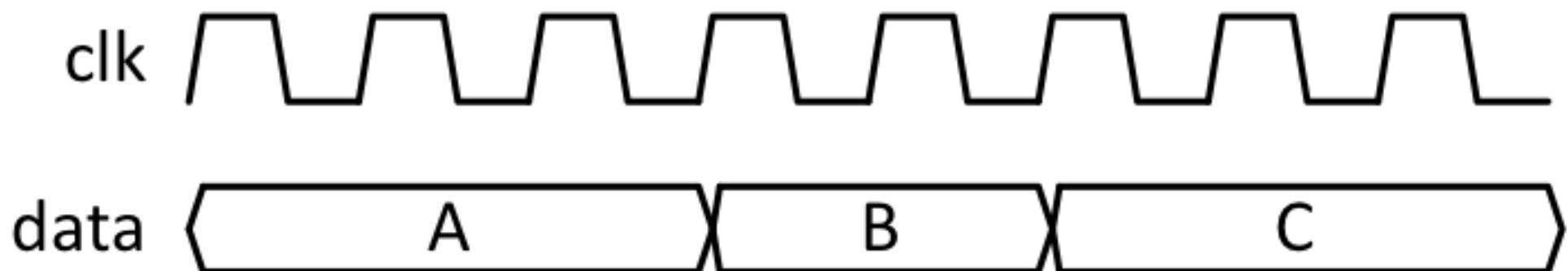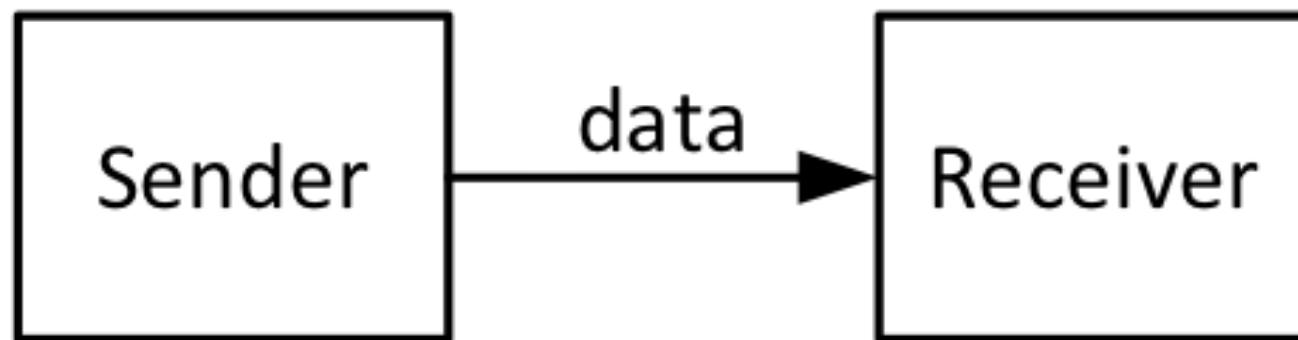
# Periodically Valid Timing
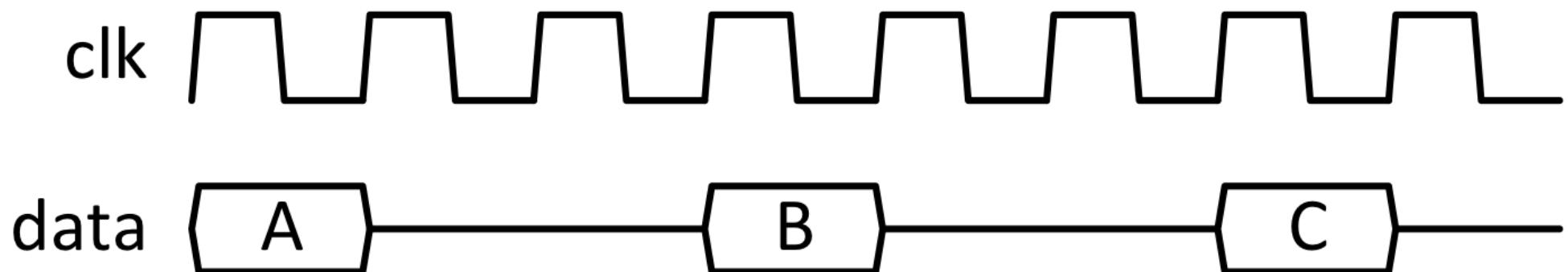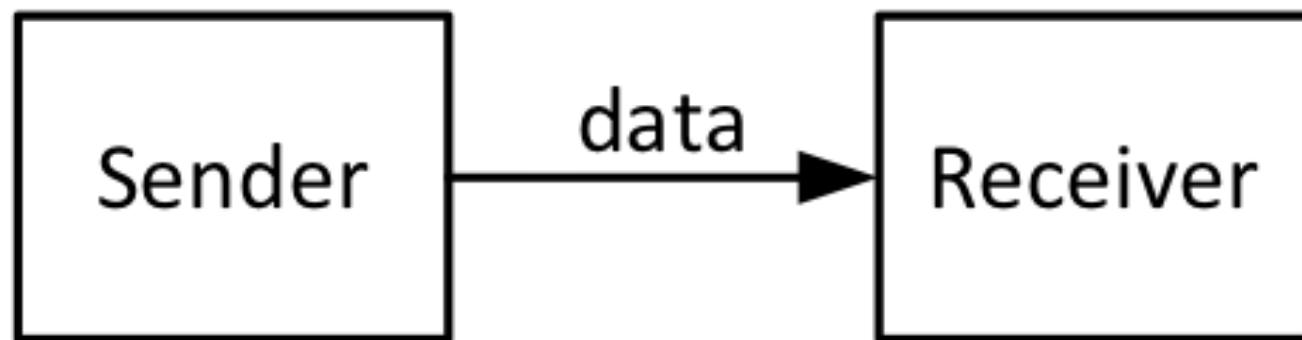
Flow Control
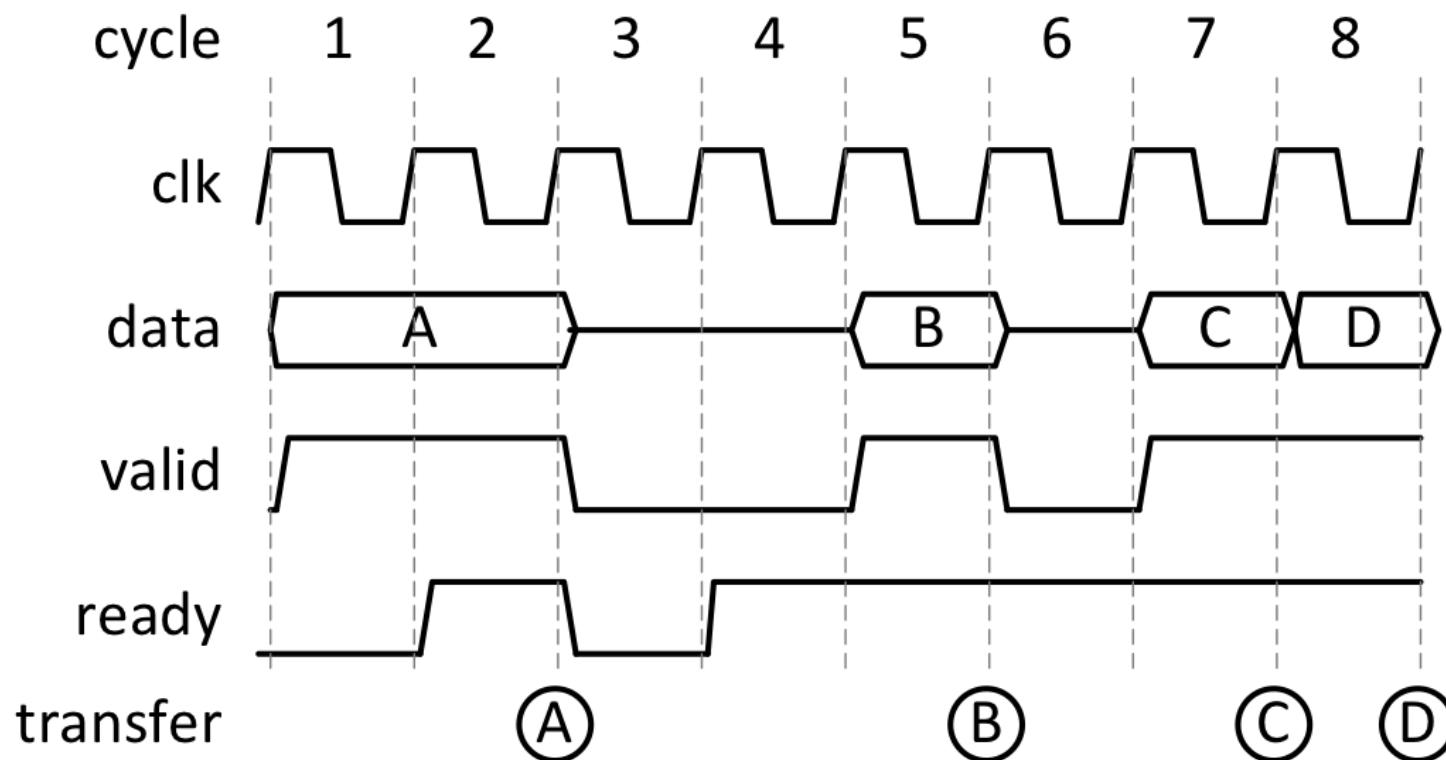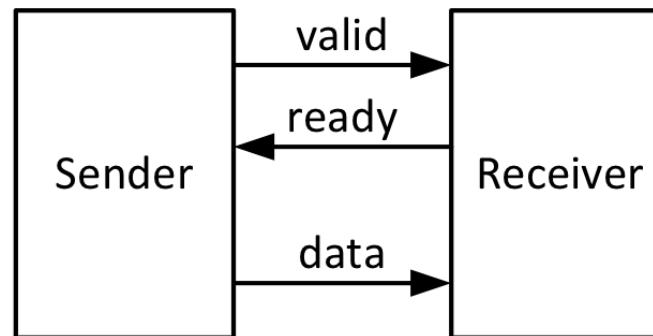
Flow-control

- Valid – Tx has data available

- Ready – Rx able to take data

- Push flow control – assume Rx always Ready

- Pull flow control – assume Tx always Valid
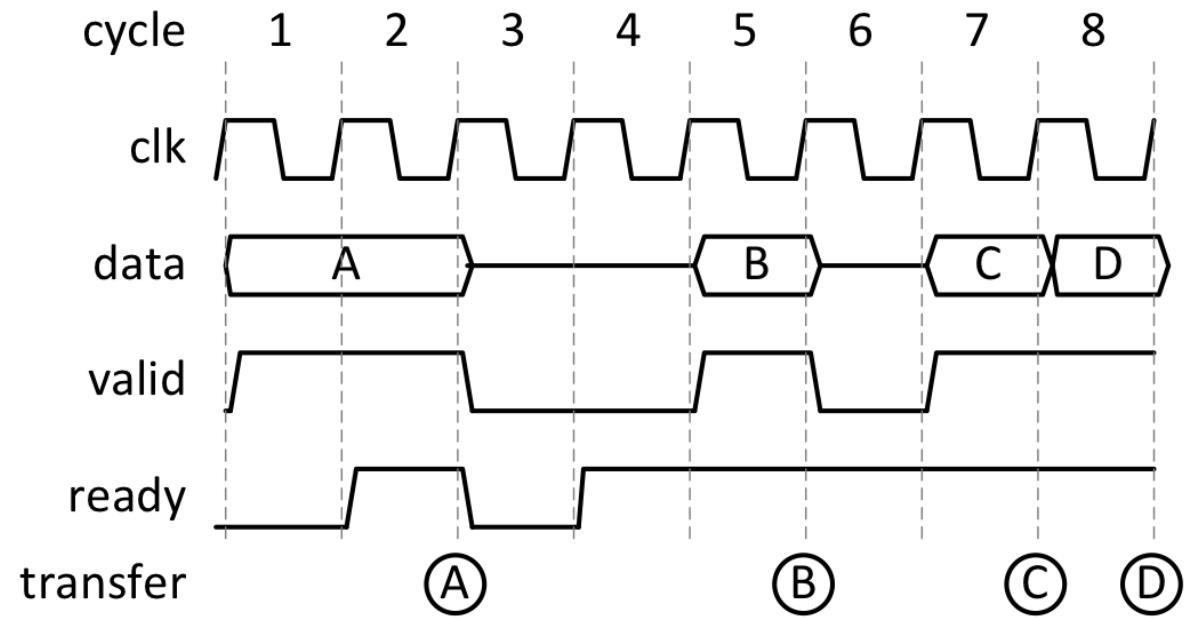
Serialization
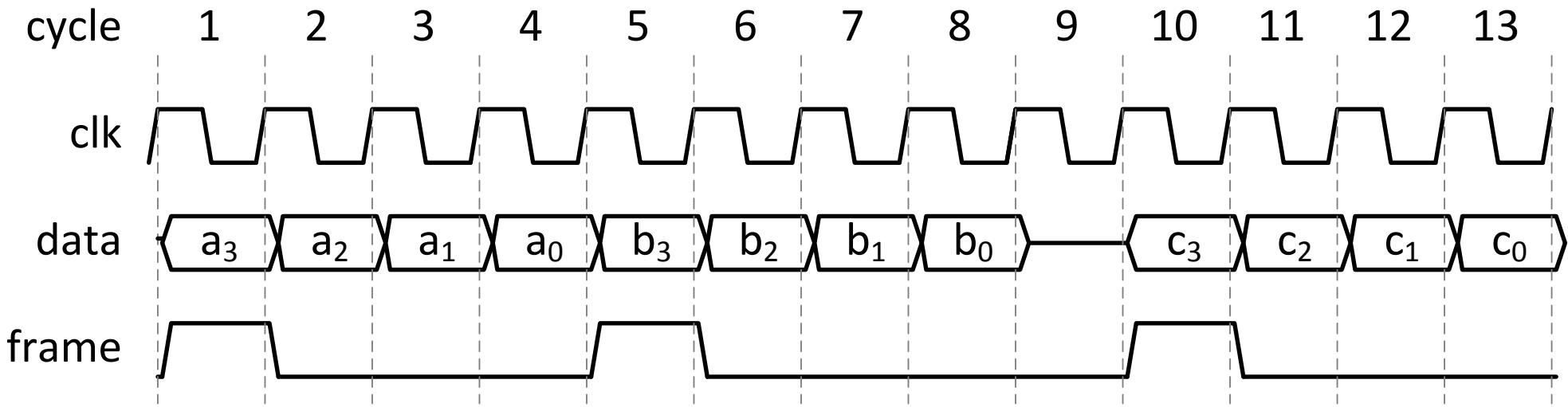
Frame signals start of new serial frame (in this case 4 words)
Flow control can be at frame granularity or word granularity

Serialization with word granularity FC
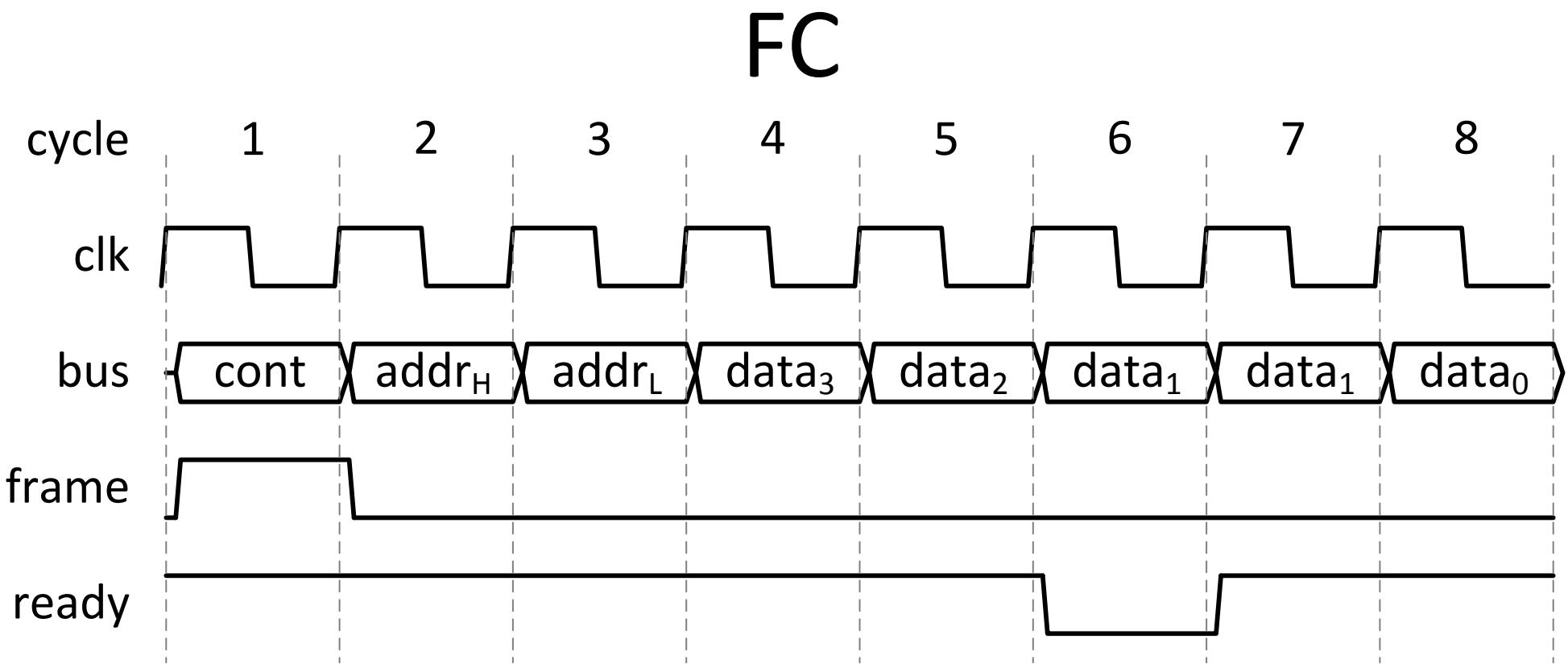
Summary

- Interface timing
  - Convention about when data is transferred
    - When valid, when accepted
    - Open loop – always valid or periodic
    - Ready-valid flow-control, both ways, push, or pull
  - Serialization

Summary of Lecture 13

- System Design
- Interfaces

- Reading
  - Book chapters 21, 22,
- Next Lecture 14:
  - Memory and interconnects