Lecture 12 SoC Design and Integration

Outline

- 12.1 SoC Bus Architecture
- 12.2 Direct Memory Access
- 12.3 RGB-to-Grayscale Converter
- 12.4 Neural Network

Outline

- 12.1 SoC Bus Architecture
- 12.2 Direct Memory Access
- 12.3 RGB-to-Grayscale Converter
- 12.4 Neural Network

- 12.1.1 AMBA AXI Bus Architecture
  - The microprocessor assumes the role of the central *manager* positioned on the high-speed AMBA AXI bus. It exercises control over a multitude of interconnected IPs.
  - These IPs, which encompass elements like the neural network engines, image/video processing units, and the floating-point (FP) hardware accelerators, operate as *subordinate* entities through the control bus, denoted by the ``S'' interface.

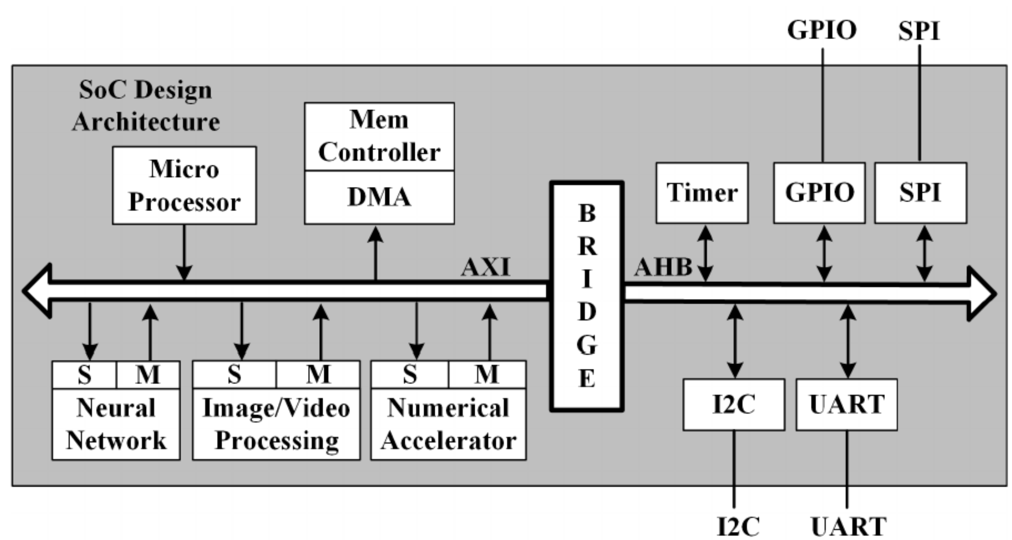


FIGURE 12.1
SoC Architecture with AMBA AXI and AHB

- 12.1.2 AMBA AXI Bus Architecture
  - Each of these IPs can also behavior as AXI *managers*, denoted as the ``M'' interface, allowing them to access memory via both the DMA and the memory controller.
  - Within this configuration, the DMA and memory controller themselves function as *subordinate* entities within the manager-subordinate structure.
    - AMBA AHB serves as the earlier version of the AMBA bus protocol and is primarily utilized for low-speed bus transfers.

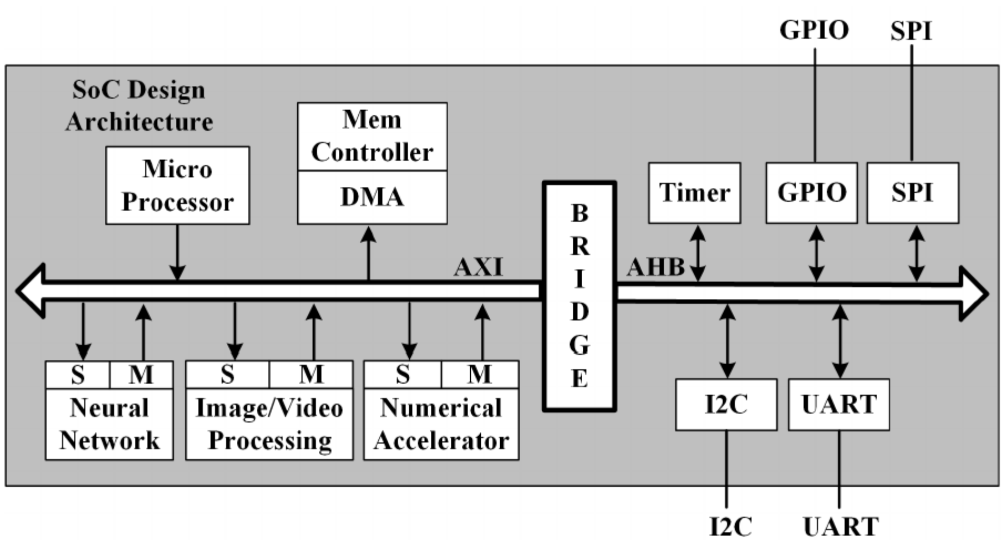


FIGURE 12.1
SoC Architecture with AMBA AXI and AHB

- 12.1.3 AMBA AXI Channels
  - The AXI protocol establishes a *point-to-point connection* between a manager and a subordinate.
  - This protocol facilitates *full-duplex communication*, allowing simultaneous read and write operations to take place on distinct channels.
  - The interaction between these two interfaces involves *five distinct channels*: Write Address (AW), Write Data (W), and Write Response (B) dedicated to write operations, and Read Address (AR) and Read Data (R) designed for read operations.

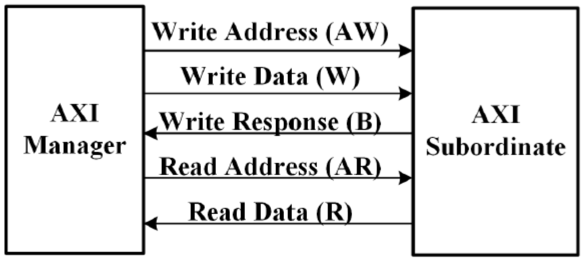


FIGURE 12.2
AMBA AXI Channels

- 12.1.3 AMBA AXI Channels
  - Write operation
    - The **manager** transmits an address through the *Write Address channel* and concurrently transfers data via the *Write Data channel* to the subordinate entity.
    - Subsequently, the **subordinate** writes the data to the specified address.
    - Following the completion of the write process, the **subordinate** transmits a response message to the manager through the *Write Response channel*.

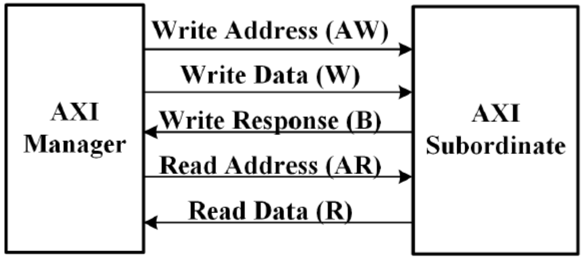


FIGURE 12.2
AMBA AXI Channels

- 12.1.3 AMBA AXI Channels
  - Read operation
    - The **manager** initiates the process by transmitting the targeted address via the *Read Address channel*.
    - The **subordinate** entity, in response, retrieves the data associated with the provided address and transmits it back to the manager through the *Read Data channel*.
      - In the event of an error, such as encountering an invalid address or lacking the necessary security permissions, the **subordinate** has the capability to communicate this error message through the *Read Data channel*.

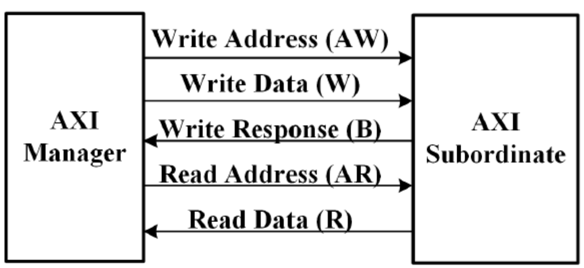


FIGURE 12.2
AMBA AXI Channels

- 12.1.4 AMBA AXI Bus Valid-Ready Handshaking
  - The **source** entity asserts the ``valid'' signal when it possesses valid information for transmission, while the **destination** entity indicates its readiness to receive information by asserting the ``ready'' signal.
  - It's important to note that the roles of source and destination can interchange depending on the specific channel in use.
    - For instance, in the *Write Address*, *Read Address*, and *Write Data channels*, the **manager** takes on the role of the **source**, while the **subordinate** assumes the role of the **destination**.
    - Conversely, in the *Write Response* and *Read Data channels*, the roles are reversed, with the **subordinate** serving as the **source** and the **manager** as the **destination**.

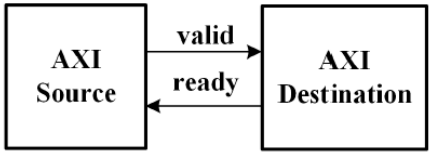
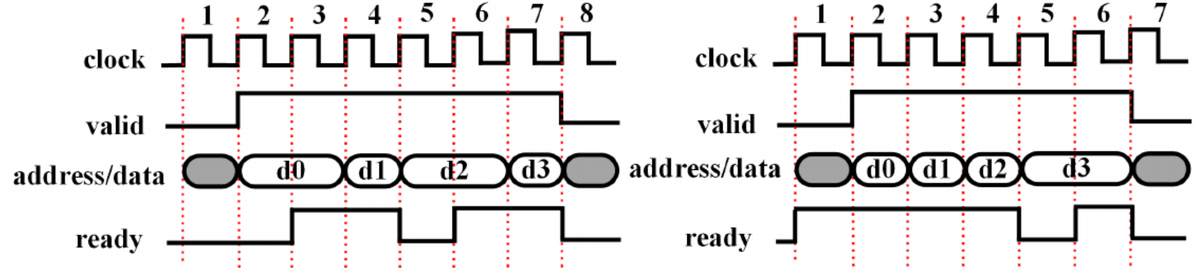


FIGURE 12.3
AMBA AXI Valid-Ready Handshaking



(a) Example 1: AMBA AXI Valid-Ready Handshaking

(b) Example 2: AMBA AXI Valid-Ready Handshaking

FIGURE 12.4
Examples of AMBA AXI Valid-Ready Handshaking

- 12.1.4 AMBA AXI Bus Valid-Ready Handshaking
  - Example 1: Handshaking
    - C2: Source asserts the ``valid'' signal to transmit ``d0'' to the destination. However, the destination isn't yet prepared to receive ``d0‘’.
    - C3: Destination asserts its ``ready'' signal, signifying its readiness to accept ``d0‘’.
    - C4: Source updates ``d1'' and asserts the ``valid'' signal. As the destination's ``ready'' signal is asserted, it becomes capable of receiving ``d1‘’.
    - C5: Source updates ``d2'' and asserts the ``valid'' signal. Since the destination isn't prepared to receive ``d2'', the source holds both the ``d2'' data and the ``valid'' signal until the following clock cycle.
    - C6: Destination is now prepared to receive ``d2‘’.
    - C7: Source updates ``d3'', and the destination promptly receives ``d3'' within the same cycle.

12.1 SoC Bus Architecture

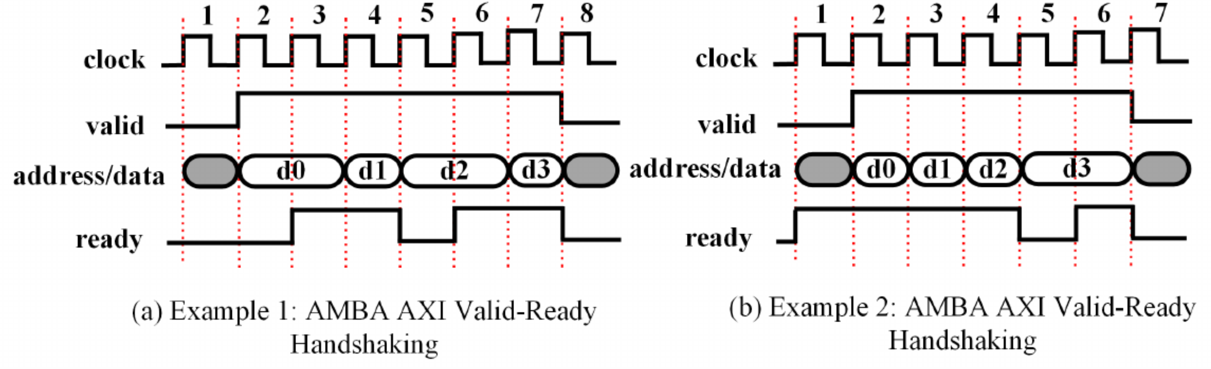
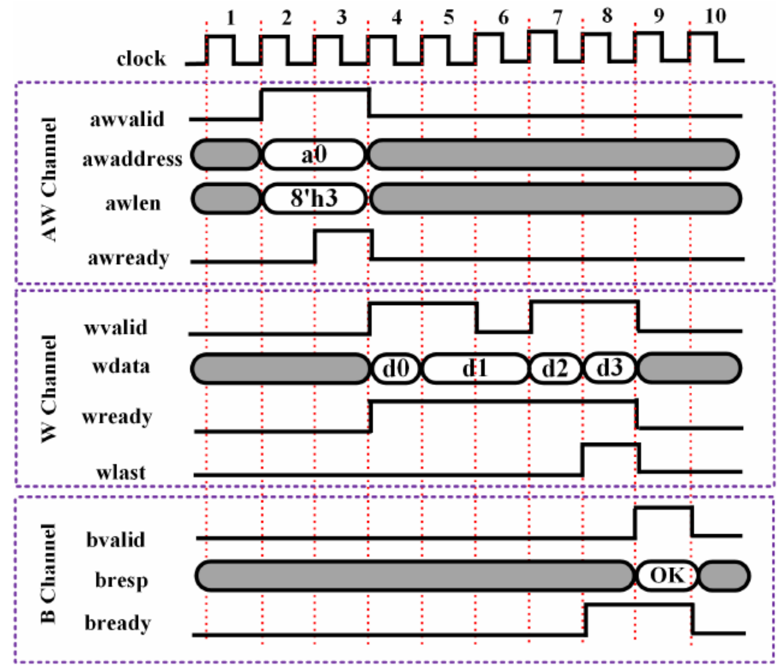


FIGURE 12.4
Examples of AMBA AXI Valid-Ready Handshaking

- 12.1.4 AMBA AXI Bus Valid-Ready Handshaking
  - Example 2: ``ready‘’ is earlier than the ``valid‘’
    - C1-4: The ``ready'' signal is asserted from cycles 1 to 4, signaling the destination's readiness to receive data. In response, the source sequentially sends ``d0'', ``d1'', and ``d2'', during cycles 2 to 4. These items are promptly received by the destination.
    - C5: Destination isn't equipped to accept further data, hence the ``ready'' signal remains unasserted. During this cycle, the source retains the ``valid'' signal and the associated ``d3'' data.
    - C6, Destination becomes capable of receiving data once more. Consequently, the last data item, ``d3'', is received by the destination. This successfully concludes all four addresses/data transfers.

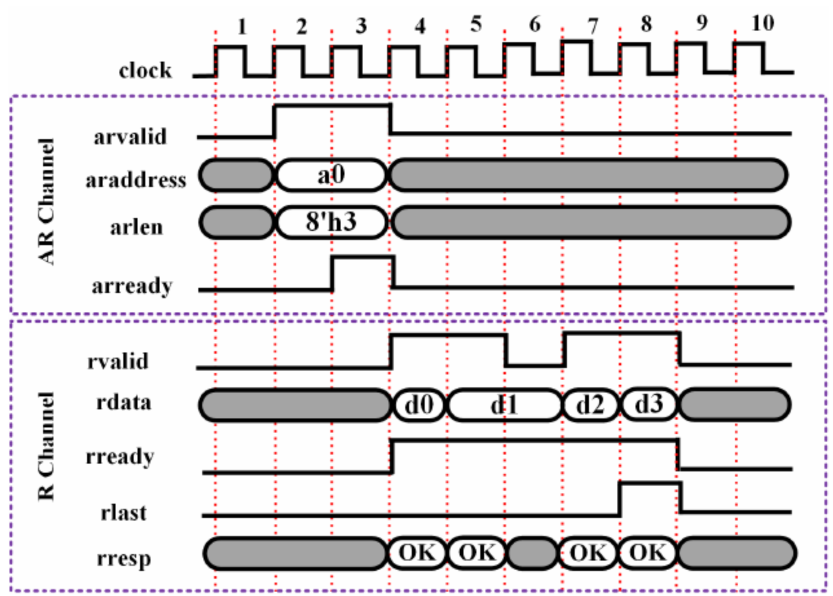
- 12.1.4 AMBA AXI Write and Read Operations
  - AXI Data Transaction and Transfer
    - Operates on a burst-based protocol
    - Five distinct transaction channels, allowing the simultaneous transfer of multiple data items within a single transaction.



(a) An Example of AMBA AXI Write Operations

  - AXI Write Operations
    - The Write Address channel initiates the process as the manager dispatches the write address (denoted as ``a0'' on the ``awaddress'' bus), the write burst length (represented as 8'h3 on the ``awlen'' bus), and other transaction-related details such as burst type and size (not shown in this figure).
    - Subsequently, the manager proceeds to transmit four data to the subordinate via the Write Data channel.
    - The Response channel comes into play as the subordinate employs it to validate the completion of the write transaction after receiving all write data.

- 12.1.4 AMBA AXI Write and Read Operations
  - AXI Read Operations
    - Read Address channel comes into play as the manager sends the read address (referred to as ``a0'' on the ``araddress'' bus) and the read burst length (represented as ``8'h3'' on the ``arlen'' bus). Alongside these details, additional commands such as burst size and type (not shown in the figure) are sent to the subordinate.
    - Subsequently, the subordinate leverages the Read Data channel to transmit four data transfers back to the manager.



(b) An Example of AMBA AXI Read Operations

FIGURE 12.5
Examples of AMBA AXI Transactions

- 12.1.5 AMBA AXI Burst Length and Burst Size
  - Burst Length
    - ``arlen[7:0]" for read transfers and ``awlen[7:0]" for write transfers

$$burst\_length = axlen + 1.$$

  - Burst Size
    - ``arsize[2:0]" for read transfers and ``awsize[2:0]" for write transfers
    - The number of data being transferred per data transaction will be:

$$trans\_size = burst\_length \times burst\_size.$$

TABLE 12.1
AXI Burst Size

axsize[2:0]	**Bytes in Each Burst**
3'b000	1
3'b001	2
3'b010	4
3'b011	8
3'b100	16
3'b101	32
3'b110	64
3'b111	128

- 12.1.6 AMBA AXI Burst Types
  - ``arburst[1:0]'' for read transfers ``awburst[1:0]'' for write transfers
  - ``axburst[1:0]'' indicates 3 burst types:
    - FIXED (axburst[1:0]==2'b00)
    - INCR (axburst[1:0]==2'b01)
    - WRAP (axburst[1:0]==2'b10)
  - Application for burst types:
    - Manager: sends control information and the address of the first data transfer in the transaction to the subordinate.
    - Subordinate: calculates the addresses of subsequent transfers in the burst. A burst must not cross a 4KB address boundary.

TABLE 12.2
AXI Burst Types

axburst[1:0]	Burst Types
2'b00	FIXED
2'b01	INCR
2'b10	WRAP
2'b11	Reserved

- 12.1.6 AMBA AXI Burst Types
  - A. AMBA AXI FIXED Burst Type
    - The address is the same for every transfer in a burst
    - E.g.: 4-beat FIXED transaction with initial address ``axaddr==32'h14", transaction length ``axlen==8’h3”, and burst size ``axsize==3'b10" or by word.
      - From the subordinate side, all the 4 beats of memory access are based on the address of 32'h14 since the burst type is FIXED.
      - This burst type is used for repeated accesses to the same location such as when loading or emptying a FIFO.

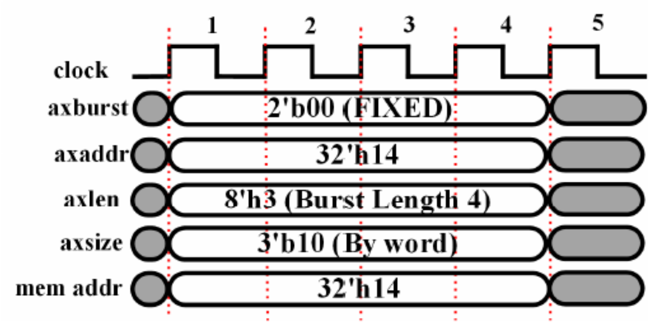
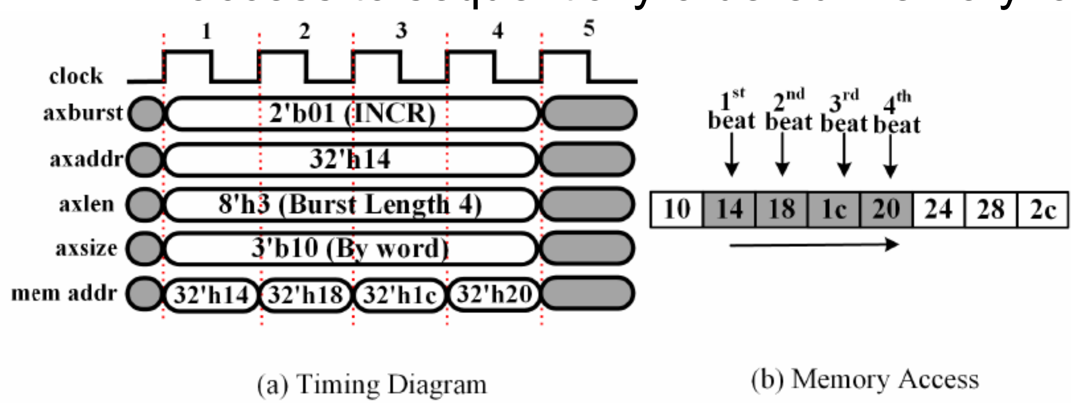


FIGURE 12.6
An Example of AMBA AXI FIXED Burst Type

TABLE 12.2
AXI Burst Types

axburst[1:0]	**Burst Types**
2’b00	FIXED
2’b01	INCR
2’b10	WRAP
2’b11	Reserved

- 12.1.6 AMBA AXI Burst Types
  - B. AMBA AXI INCR Burst Type
    - The address for each transfer in the burst is an increment of the address for the previous transfer
    - E.g. 4-beat INCR transaction, with initial address ``axaddr==32'h14‘’, transaction length ``axlen==8’h3‘’, and burst size ``axsize==3'b10" or by word
      - This progression yields a sequence of memory addresses: 32'h14, 32'h18, 32'h1c, and 32’h20
      - This burst type finds common application in scenarios necessitating access to sequentially ordered memory locations.



(a) Timing Diagram

(b) Memory Access

FIGURE 12.7
An Example of AMBA AXI INCR Burst Type

TABLE 12.2
AXI Burst Types

axburst[1:0]	Burst Types
2’b00	FIXED
2’b01	INCR
2’b10	WRAP
2’b11	Reserved

- 12.1.6 AMBA AXI Burst Types
  - C. AMBA AXI WRAP Burst Type
    - The address cycle returns to the *foundational address* when the higher address value coincides with the wrap boundary.

$$base\_addr = \{axaddr[MSB-1:N], N'b0\}.$$

N signifies the bit width of the data transaction size, $2^N = trans\_size.$

- The *wrap boundary* can be further articulated as:

$$wrap\_boundary = base\_address + trans\_size.$$

TABLE 12.1
AXI Burst Size

axsize[2:0]	Bytes in Each Burst
3'b000	1
3'b001	2
3'b010	4
3'b011	8
3'b100	16
3'b101	32
3'b110	64
3'b111	128

TABLE 12.2
AXI Burst Types

axburst[1:0]	Burst Types
2'b00	FIXED
2'b01	INCR
2'b10	WRAP
2'b11	Reserved

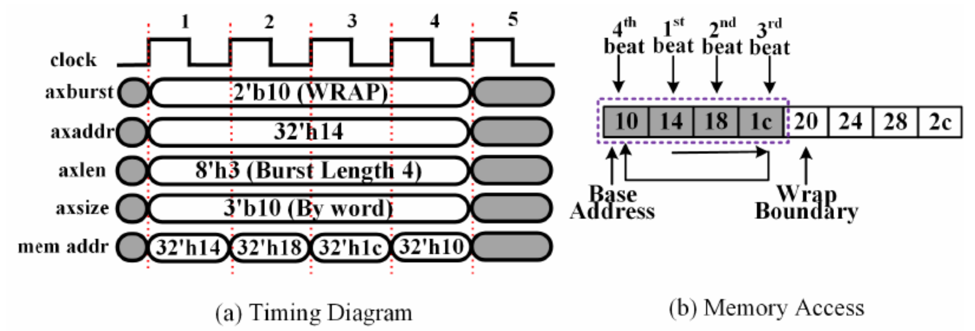


FIGURE 12.8
An Example of AMBA AXI WRAP Burst Type

- 12.1.6 AMBA AXI Burst Types
  - C. AMBA AXI WRAP Burst Type
    - Example:
      - Four-beat WRAP transaction: address is ``axaddr=32'h14'', the burst length is ``axlen=8'h3'', corresponding to four data bursts, and the burst size is ``axsize=3'b10'', indicating word-sized bursts.
      - Consequently, the transaction size can be computed as

burst_size x burst_length = 16 bytes (in hexadecimal 32'h10)

So N=4 (due to 2^4=16) $base\_addr = \{axaddr[MSB-1:N], N'b0\}.$

Therefore, substituting the four least significant bits with zeros yields the base address as 32’h10

Furthermore, the wrap boundary is established as

base_address + trans_size=32'h10+32'h10=32'h20.

12.1 SoC Bus Architecture

- 12.1.6 AMBA AXI Burst Types
  - C. AMBA AXI WRAP Burst Type

TABLE 12.3
Base Address for Wrap Burst Type

Burst Length	Burst Size	N	Base Address
2	1	1	{axaddr[MSB-1:1], 1'b0}
	2	2	{axaddr[MSB-1:2], 2'b0}
	4	3	{axaddr[MSB-1:3], 3'b0}
	8	4	{axaddr[MSB-1:4], 4'b0}
	16	5	{axaddr[MSB-1:5], 5'b0}
	32	6	{axaddr[MSB-1:6], 6'b0}
	64	7	{axaddr[MSB-1:7], 7'b0}
	128	8	{axaddr[MSB-1:8], 8'b0}
4	1	2	{axaddr[MSB-1:2], 2'b0}
	2	3	{axaddr[MSB-1:3], 3'b0}
	4	4	{axaddr[MSB-1:4], 4'b0}
	8	5	{axaddr[MSB-1:5], 5'b0}
	16	6	{axaddr[MSB-1:6], 6'b0}
	32	7	{axaddr[MSB-1:7], 7'b0}
	64	8	{axaddr[MSB-1:8], 8'b0}
	128	9	{axaddr[MSB-1:9], 9'b0}
8	1	3	{axaddr[MSB-1:3], 3'b0}
	2	4	{axaddr[MSB-1:4], 4'b0}
	4	5	{axaddr[MSB-1:5], 5'b0}
	8	6	{axaddr[MSB-1:6], 6'b0}
	16	7	{axaddr[MSB-1:7], 7'b0}
	32	8	{axaddr[MSB-1:8], 8'b0}
	64	9	{axaddr[MSB-1:9], 9'b0}
	128	10	{axaddr[MSB-1:10], 10'b0}
16	1	4	{axaddr[MSB-1:4], 4'b0}
	2	5	{axaddr[MSB-1:5], 5'b0}
	4	6	{axaddr[MSB-1:6], 6'b0}
	8	7	{axaddr[MSB-1:7], 7'b0}
	16	8	{axaddr[MSB-1:8], 8'b0}
	32	9	{axaddr[MSB-1:9], 9'b0}
	64	10	{axaddr[MSB-1:10], 10'b0}
	128	11	{axaddr[MSB-1:11], 11'b0}

- 12.1.7 AMBA AXI Address Alignment
  - Memory accessed via the AXI bus necessitate address alignment in accordance with the designated burst sizes.

$$aligned\_addr = INT(\frac{start\_addr}{burst\_size}) \times burst\_size.$$

- E.g.
  - A manager issues a command with transfer type ``axburst=2'b01" (indicating an INCR transaction), a burst length of ``axlen=8'h3" (burst_length=4), and a burst size of ``axsize=3'b10" (burst_size=4).
  - In the scenario where the start address is erroneously set to 0x11 by software, leading to misalignment with the burst size, hardware will rectify the address to 32'h10 using the equation

$$aligned\_addr = INT\left(\frac{17}{4}\right) \times 4 = 16$$

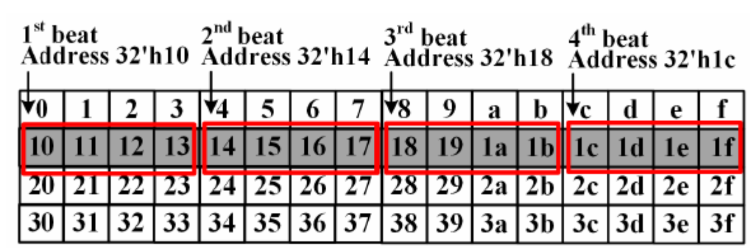


FIGURE 12.9
An Example of Address Alignment

Outline

- 12.1 SoC Bus Architecture
- 12.2 Direct Memory Access
- 12.3 RGB-to-Grayscale Converter
- 12.4 Neural Network

- 12.2.1 DMA Design Architecture
  - Determined by priority assignments, the arbiter resolves access allocation, permitting engagement with a single master at a time.

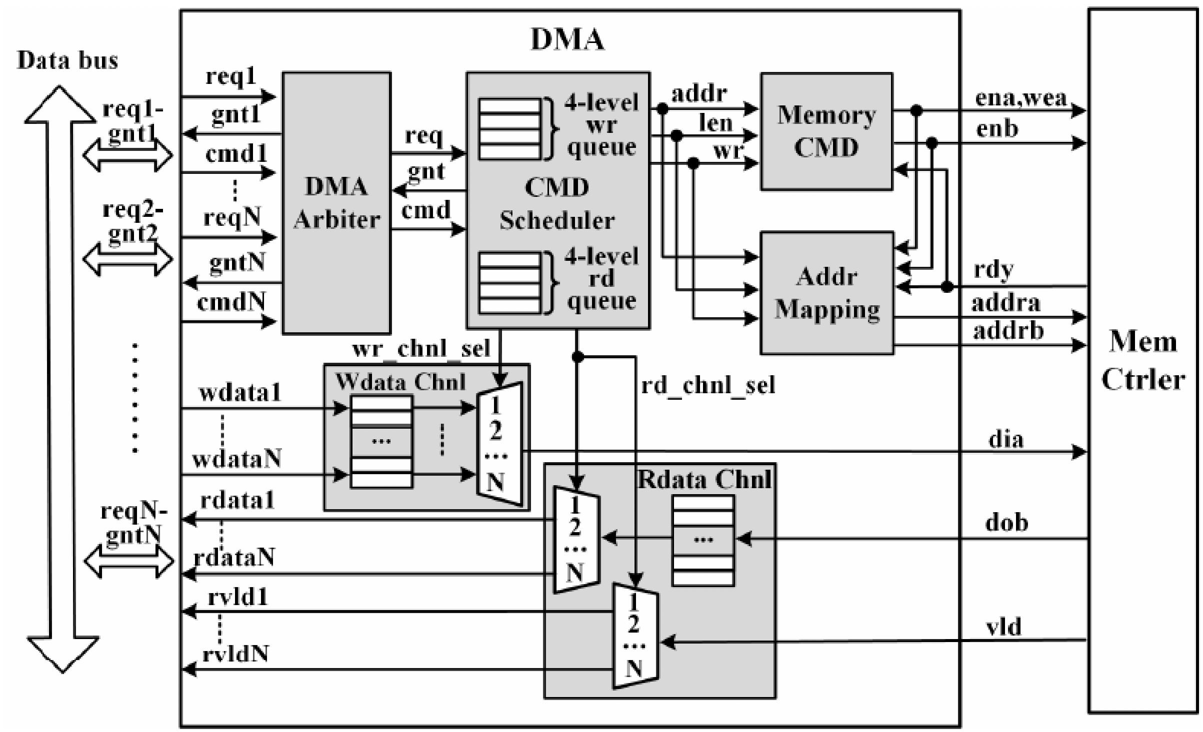


FIGURE 12.10
DMA Design Architecture

- 12.2.1 DMA Arbiter Design
- A. DMA Arbitration

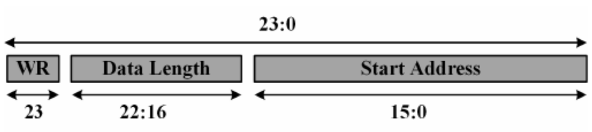


FIGURE 12.12
DMA Command

Time Slots	M1 Priority No.	M2 Priority No.	M3 Priority No.	M4 Priority No.
1	1	2	3	4
2	2	4	6	4
3	3	6	3	8
4	4	8	6	4
5	5	2	9	8
6	6	4	3	12
7	7	6	6	4
8	1	8	9	8

FIGURE 12.11
Dynamic Priority Mechanism on DMA Arbitration

- B. DMA Arbiter IOs
- C. Timing Diagram of DMA Arbiter

TABLE 12.4
DMA Arbiter IOs

Names	Width	IOs	Description
req1∼req4	1	Input	Bus requests from M1∼M4
cmd1∼cmd4	24	Input	Bus commands from M1∼M4
gnt1∼gn4	1	Output	Bus grants for M1∼M4
req	1	Output	Command scheduler request
cmd	24	Output	Command scheduler command
gnt	1	Input	Command scheduler grant

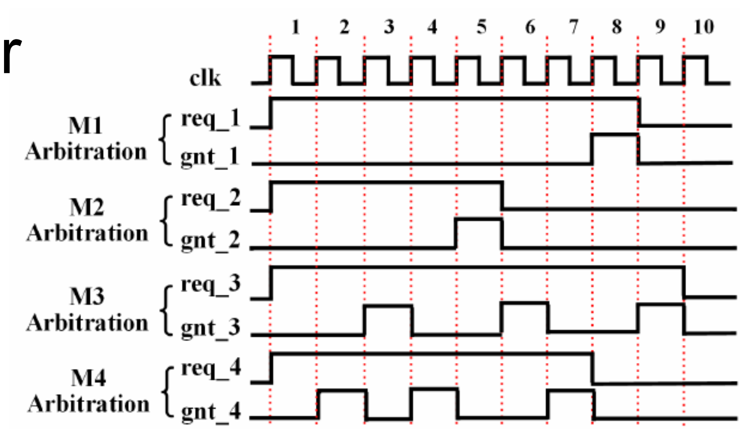


FIGURE 12.13
Timing Diagram of DMA Arbitration

- 12.2.3 DMA Command and Address Mapping
  - A. Memory Command and Address Mapping IOs
  - B. Timing Diagram of Memory Command and Address Mapping

TABLE 12.5
Memory Command and Address Mapping IOs

Names	Width	IOs	Description
wr	1	Input	DMA write/read indicator: 1'b1 for write, 1'b0 for read
len	7	Input	DMA write data length
addr	16	Input	DMA write address
ena	1	Output	Memory port "a" enable
wea	4	Output	Memory write enable to each data byte
enb	1	Output	Memory port "b" enable
rdy	2	Input	Memory ready response: MSB for write, LSB for read
addra	16	Output	Memory port "a" address
addrb	16	Output	Memory port "b" address

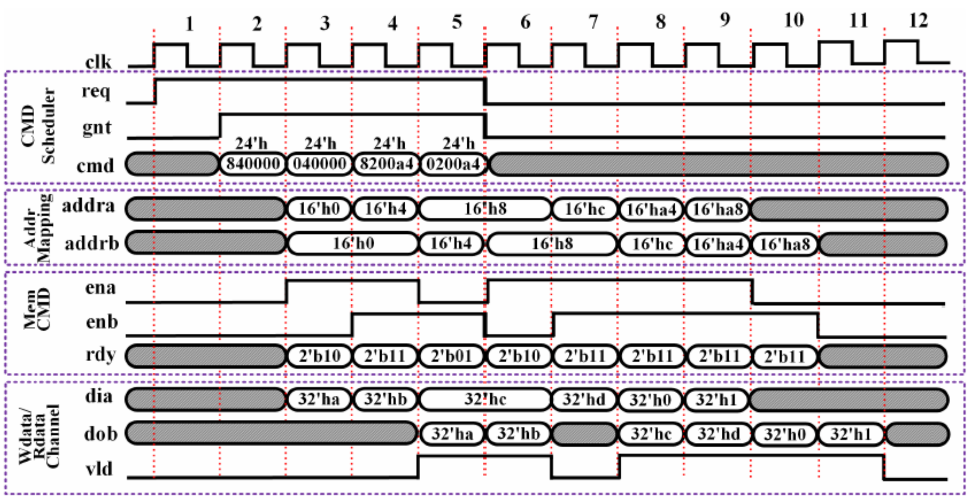


FIGURE 12.14
Timing Diagram of Memory Command and Address Mapping

Outline

- 12.1 SoC Bus Architecture
- 12.2 Direct Memory Access
- 12.3 RGB-to-Grayscale Converter
- 12.4 Neural Network

- 12.3.1 Floating-Point Design
  - A. Grayscale and Color Images
    - Grayscale images, pixels convey only intensity information, ranging from black (weakest intensity) to white (strongest intensity).
      - A black pixel can be represented by the 8-bit value 8'h0, while a white pixel corresponds to the 8-bit value 8'hff.
      - To define the ***image grid***, we simply multiply the width by the height of the pixel matrix. This grid is commonly referred to as the image's resolution.
        - » When we say ``640 x 480'', it indicates an image resolution with 640 columns and 480 rows.
    - In a format like RGB888, a single color pixel (red, green, or blue) is composed of 8 bits of data, with all three components utilizing 8 bits each.
      - For example, {8'h0, 8'h0, 8'hff} or 24'hff signifies a blue pixel, with blue set to 8'hff while red and green are set to 8'h0.
      - Comparing this to a one-dimensional grayscale image, a color image using the RGB888 format can be visualized as a three-dimensional grid or array, often referred to as a ``*tensor*''. The three dimensions of this tensor represent the width, height, and the image's color channels.

- 12.3.1 Floating-Point Design
  - B. RGB-to-Grayscale Conversion Algorithm

$$grayscale = 0.2989 \times r + 0.587 \times g + 0.114 \times b$$

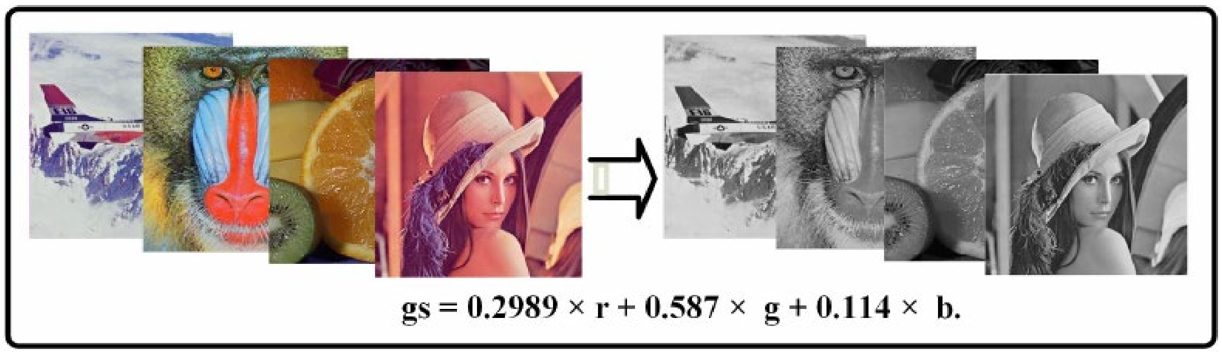


FIGURE 12.15
RGB-to-Grayscale Conversion

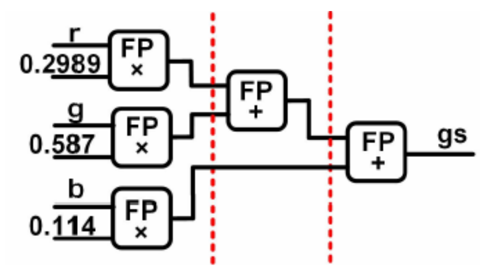


FIGURE 12.16
Floating-Point Design on RGB-to-Grayscale Converter

12.3.2 Floating-to-Integer Conversion and Approximate Designs

- 12.3.2 Floating-to-Integer Conversion and Approximate Designs
  - A. Floating-to-Fixed-Point Conversion
    - The fixed-point representation is primarily employed to portray integer values, distinct from FP numbers.
      - For instance, to represent a single precision number like 3.14159, the number can be scaled by 2^16 (decimal 65536) to generate the integer value 205887, which is mathematically represented and approximated as $3.14159 \times 65536 = 205887.67264 \approx 205887$. Subsequently, this integer value is encoded using multiple bits of binary data to depict the FP number 3.14159.
      - Nevertheless, it's imperative to recognize that this technique involves a *compromise in precision* in order to render the FP number as an integer.
      - While integer designs tend to exhibit *enhanced power efficiency and cost-effectiveness* for certain applications, they might not deliver the exact *accuracy* and *quality of results* required.

- 12.3.2 Floating-to-Integer Conversion and Approximate Designs
  - B. Integer Design \#1: Floating-to-Integer Conversion

$$
\begin{aligned}
gs &= 0.2989 \times r + 0.5870 \times g + 0.1140 \times b \\
&= [2^7 \times (0.2989 \times r + 0.5870 \times g + 0.1140 \times b)]/2^7 \\
&\approx (38 \times r + 75 \times g + 15 \times b)/2^7
\end{aligned}
$$

  - C. Integer Design #2: Integer Adder Based Design

$$
\begin{aligned}
gs &= 0.2989 \times r + 0.5870 \times g + 0.1140 \times b \\
&= 2^5 \times (0.2989 \times r + 0.5870 \times g + 0.1140 \times b)/2^5 \\
&\approx (10 \times r + 19 \times g + 4 \times b)/2^5 \\
&= (2^3 \times r + 2 \times r) + (2^4 \times g + 2 \times g + g) + (2^2 \times b)
\end{aligned}
$$

12.3.2 Floating-to- Integer Conversion and Approximate Designs

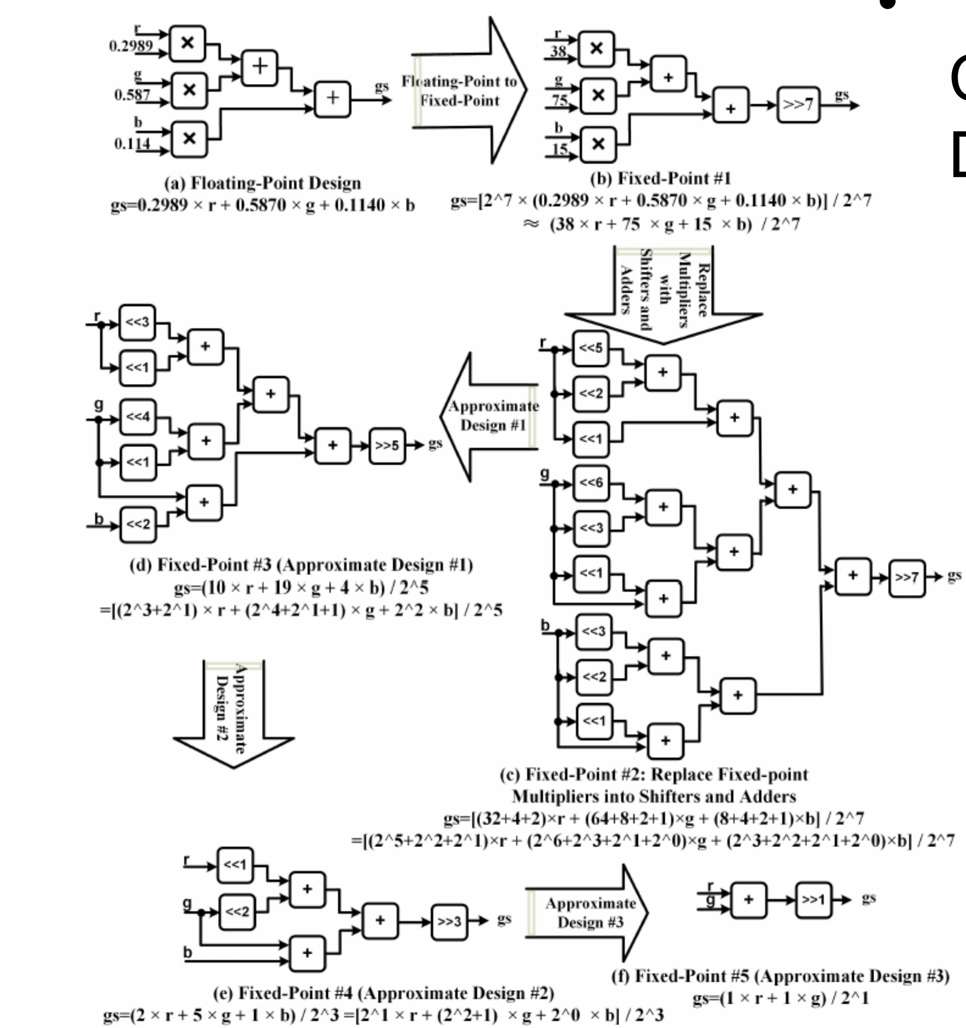


FIGURE 12.17
Floating-to-Fixed-Point Designs of RGB-to-Grayscale Converter

- 12.3.2 Floating-to-Integer Conversion and Approximate Designs
  - D. Integer Design #3-5: Approximate Designs

$$
\begin{aligned}
gs &= 0.2989 \times r + 0.5870 \times g + 0.1140 \times b \\
&= 2^5 \times (0.2989 \times r + 0.5870 \times g + 0.1140 \times b)/2^5 \\
&\approx (10 \times r + 19 \times g + 4 \times b)/2^5 \\
&= (2^3 \times r + 2 \times r) + (2^4 \times g + 2 \times g + g) + (2^2 \times b)
\end{aligned}
$$

$$
\begin{aligned}
gs &= 0.2989 \times r + 0.5870 \times g + 0.1140 \times b \\
&= 2^3 \times (0.2989 \times r + 0.5870 \times g + 0.1140 \times b)/2^3 \\
&\approx (2 \times r + 5 \times g + 1 \times b)/2^3 \\
&= (2 \times r) + (2^2 \times g + g) + b
\end{aligned}
$$

$$
\begin{aligned}
gs &= 0.2989 \times r + 0.5870 \times g + 0.1140 \times b \\
&= 2^1 \times (0.2989 \times r + 0.5870 \times g + 0.1140 \times b)/2^1 \\
&\approx 1 \times r + 1 \times g + 0 \times b/2^1 \\
&= (r + g)/2
\end{aligned}
$$

12.3.3 Hardware Utilization vs. Speed Estimation

- 12.3.3 Hardware Utilization vs. Speed Estimation

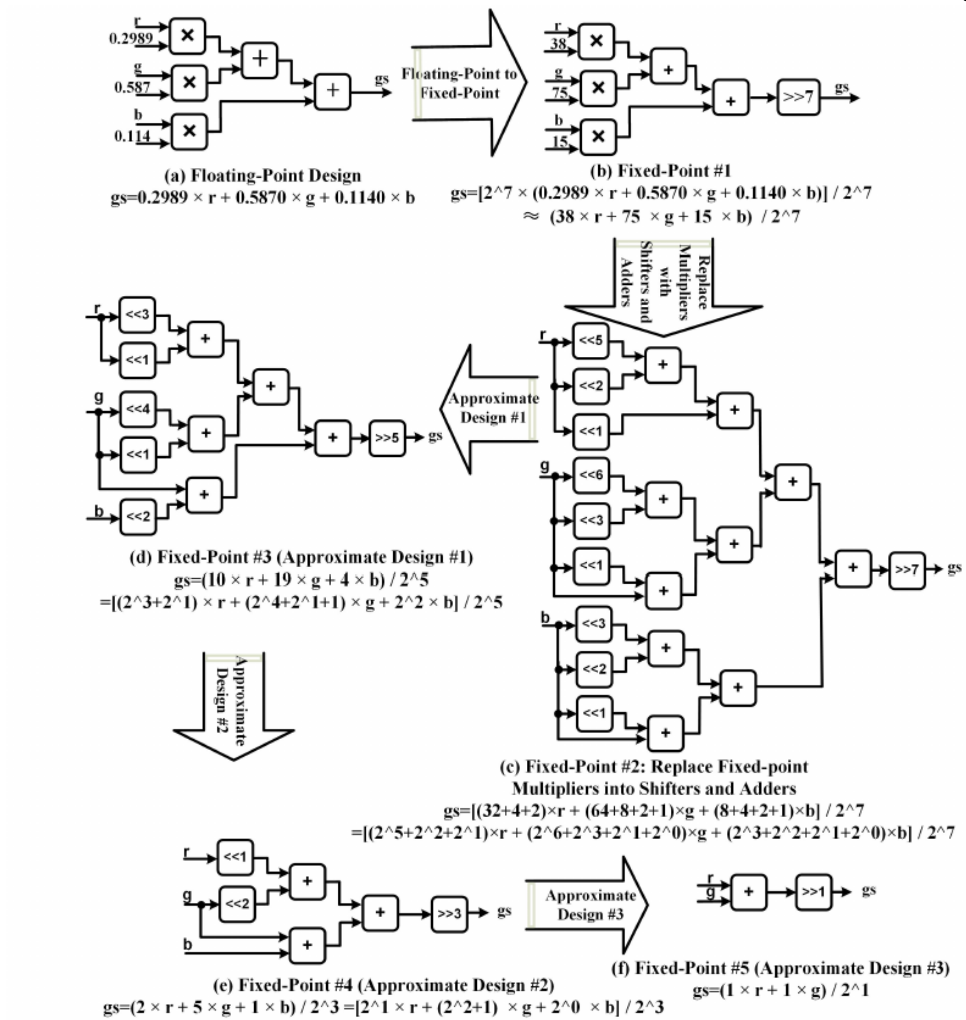


FIGURE 12.17
Floating-to-Fixed-Point Designs of RGB-to-Grayscale Converter

TABLE 12.6
Resource Utilization vs. Speed of Fixed-Point Designs

Designs	Hardware Cost	Critical Path
Design#1	3 Multiplier+2 Adders	1 Multiplier + 2 Adders
Design#2	10 Adders	4 Adders
Design#3	5 Adders	3 Adders
Design#4	3 Adders	2 Adders
Design#5	1 Adder	1 Adder

Outline

- 12.1 SoC Bus Architecture
- 12.2 Direct Memory Access
- 12.3 RGB-to-Grayscale Converter
- 12.4 Neural Network

12.4.1 Single-Layer Perceptron Neural Network

- 12.4.1 Single-Layer Perceptron Neural Network
  - A. Design Structure of SLP Neural Network
  - B. SLP Sigmoid Neuron

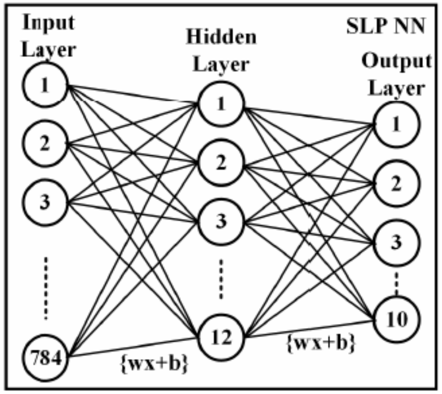


FIGURE 12.18
Design Structure of SLP Neural Network

$$hn_i = \frac{1}{1 + exp(-\sum_{j=1}^{784} wh_{ij} \times x_j - bh_i)}$$

$$on_i = \frac{1}{1 + exp(-\sum_{j=1}^{12} wo_{ij} \times x_j - bo_i)}$$

- 12.4.2 Streaming Design on SLP Neural Network
  - Streaming Design on Sigmoid Neurons

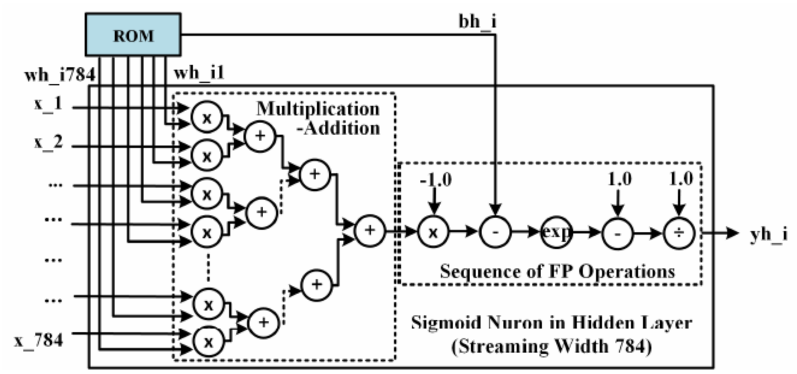


FIGURE 12.19
Streaming Design Structure of Hidden-Layer Neuron

$$hn_i = \frac{1}{1 + exp(-\sum_{j=1}^{784} wh_{ij} \times x_j - bh_i)}$$

$$on_i = \frac{1}{1 + exp(-\sum_{j=1}^{12} wo_{ij} \times x_j - bo_i)}$$

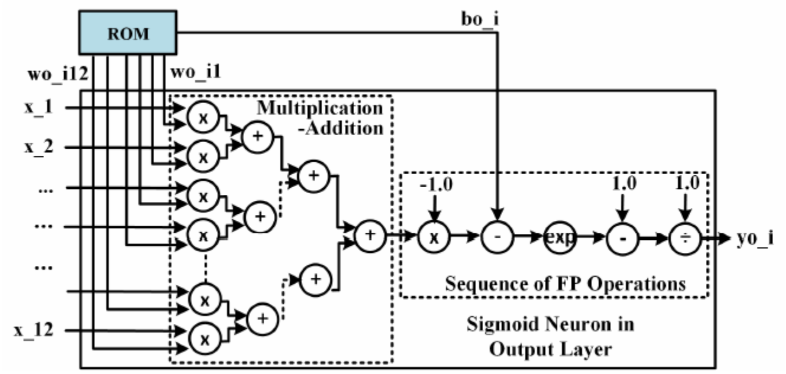


FIGURE 12.20
Streaming Design Structure of Output-Layer Neuron

- 12.4.2 Streaming Design on SLP Neural Network
  - Timing Diagram of Streaming Design on SLP Neural Network

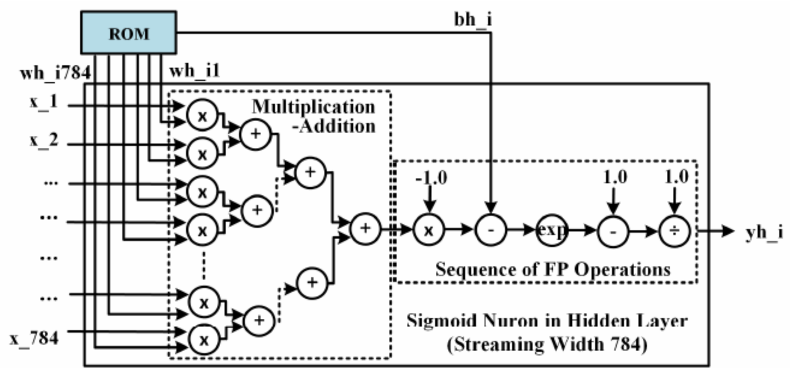


FIGURE 12.19
Streaming Design Structure of Hidden-Layer Neuron

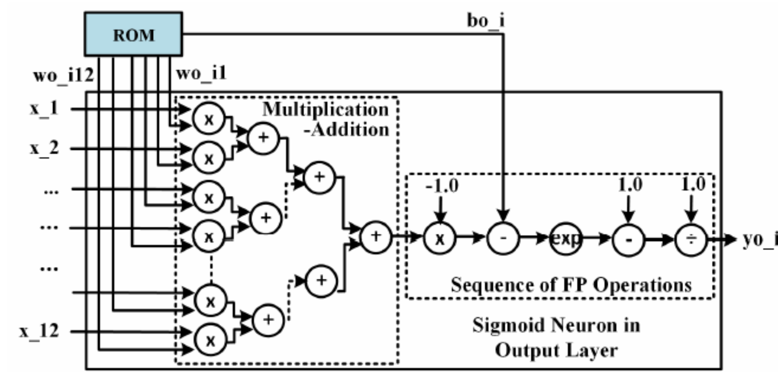


FIGURE 12.20
Streaming Design Structure of Output-Layer Neuron

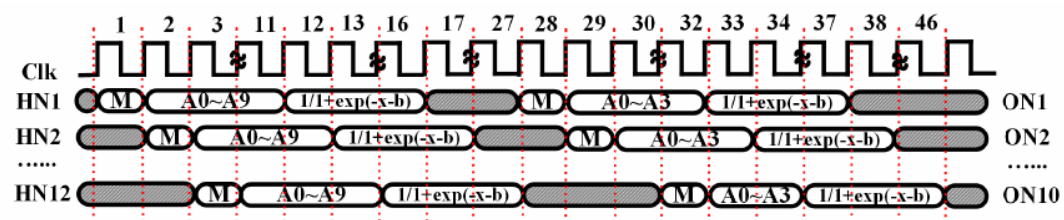


FIGURE 12.21
Timing Diagram of Steaming Design on SLP Neural Network

12.4.3 Iterative Design on SLP Neural Network

- 12.4.3 Iterative Design on SLP Neural Network
  - Iterative Design on Sigmoid Neurons

$$hn_i = \frac{1}{1 + exp(-\sum_{j=1}^{784} wh_{ij} \times x_j - bh_i)}$$

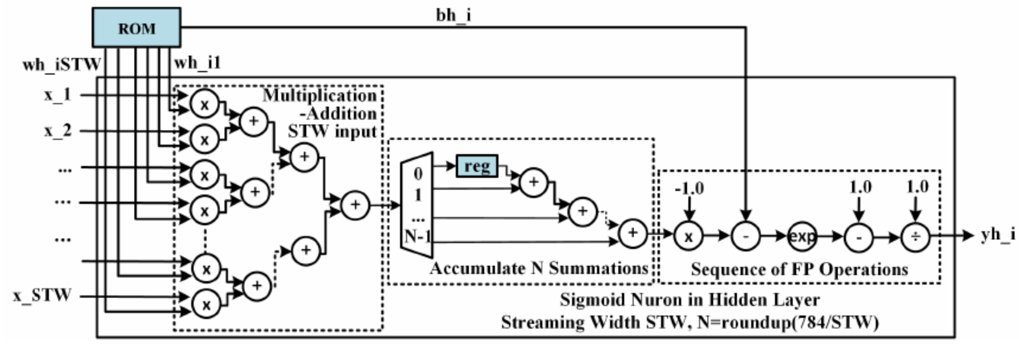


FIGURE 12.22
Iterative Design Structure of Hidden-Layer Neuron

  - Latency Assessment of Hidden-Layer Sigmoid Neurons in Iterative Design

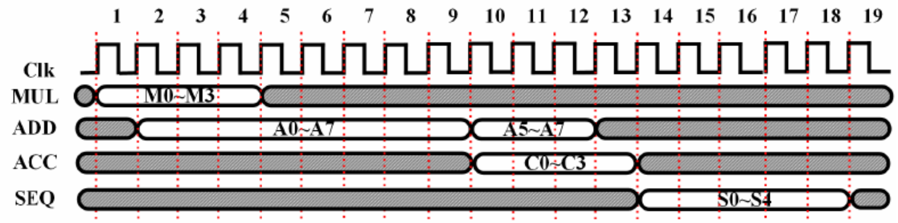


FIGURE 12.23
Timing Diagram of Hidden-Layer Sigmoid Neuron within Iterative Design

- 12.4.2 Iterative Design on SLP Neural Network
  - Timing Diagram of Iterative Design on SLP Hidden-Layer Neural Network
    - A. Iterative Design with Streaming Width 196

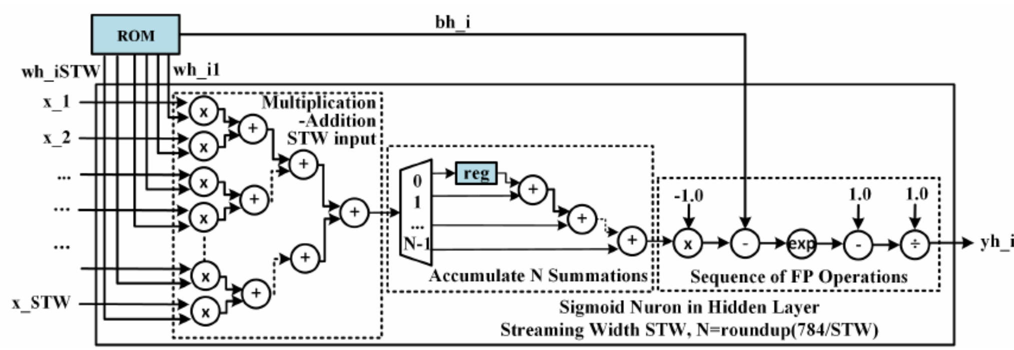


FIGURE 12.22
Iterative Design Structure of Hidden-Layer Neuron

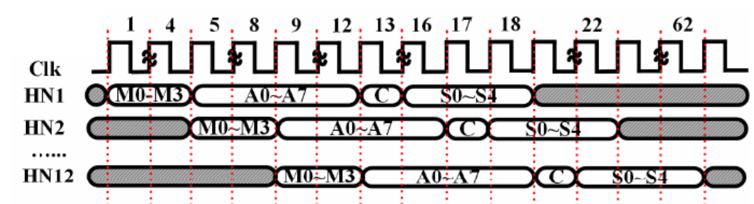


FIGURE 12.24
Timing Diagram of Iterative Design (Streaming Width 196) on Hidden-Layer Sigmoid Neurons

- 12.4.2 Iterative Design on SLP Neural Network
  - Timing Diagram of Iterative Design on SLP Hidden-Layer Neural Network
    - B. Iterative Design with Streaming Width 98

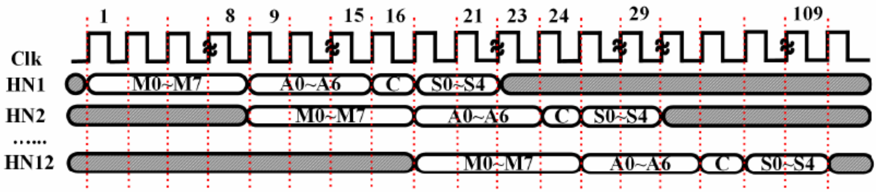
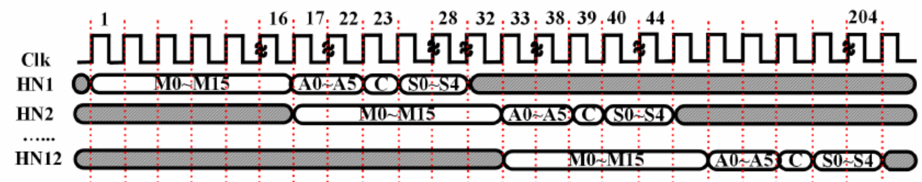
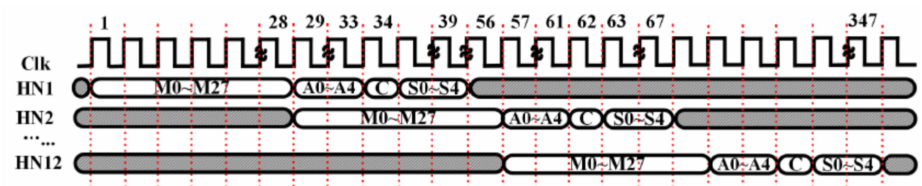


FIGURE 12.25
Timing Diagram of Iterative Design (Streaming Width 98) on Hidden-Layer Sigmoid Neurons



(a) Timing Diagram of Iterative Design with Streaming Width 49



(b) Timing Diagram of Iterative Design with Streaming Width 28

FIGURE 12.26
Timing Diagram of Iterative Design (Streaming Widths 49 and 28) on Hidden-Layer Sigmoid Neurons

- C. Iterative Designs with Streaming Widths 49 and 28

- 12.4.2 Iterative Design on SLP Neural Network
  - Hardware Utilization vs. Latency Estimation
    - A. Design Performance of Hidden-Layer Neurons

TABLE 12.7
Resource Cost vs. Latency of Hidden-Layer Neurons

NNs-STW	Hardware Cost					Latency
	MUL	ADD	SUB	EXP	REC	/(Cycles)
784	785	783	2	1	1	27
196	197	198	2	1	1	62
98	99	104	2	1	1	109
49	50	63	2	1	1	204
28	29	54	2	1	1	347

    - B. Design Performance of SLP Neural Networks

TABLE 12.8
Resource Cost vs. Latency of SLP Neural Networks

NNs-STW	Hardware Cost					Latency
	MUL	ADD	SUB	EXP	REC	/(Cycles)
784	798	794	4	2	2	46
196	210	209	4	2	2	81
98	112	115	4	2	2	128
49	63	74	4	2	2	223
28	42	65	4	2	2	366

Thanks!

CRC Publisher Book: Integrated Circuit Design and Simulation (in 2024)

Associate Professor, UHCL

yangxia@uhcl.edu

Affiliate Faculty, LBL

xiaokunyang@lbl.gov