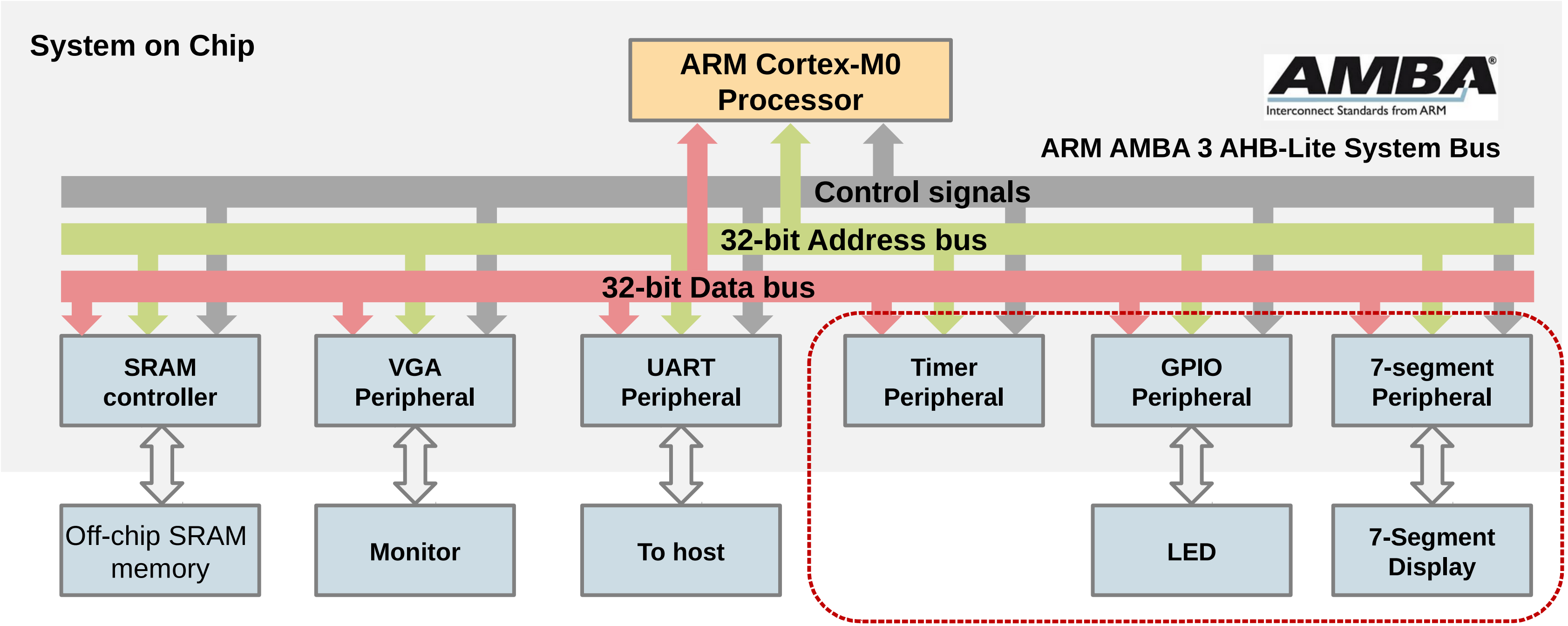
Design and Implementation of Timer, GPIO, and 7-segment Peripherals

Module Overview

- Learn about timers, GPIO and 7-segment display;
- Design and implement an AHB timer, a GPIO peripheral, and a 7-segment display peripheral;
- Program peripherals using assembly;
- Lab Demonstration.



Module Syllabus

- Timer Overview
- Components of a Standard Timer
- Timer Operation Mode
- AHB Timer Implementation
- GPIO Overview
- AHB GPIO Implementation
- 7-Segment Display Overview
- AHB 7-Segment Display Implementation
- Lab Practice

Timer

Timer Overview

- A hardware timer is basically a digital counter that:
  - Counts regular events, which normally refers to a clock source that has a relatively high, and fixed frequency;
  - Increment or decrement at a fixed frequency;
  - Resets itself when reaching zero, or a predefined value;
  - Generates an interrupt when reset;

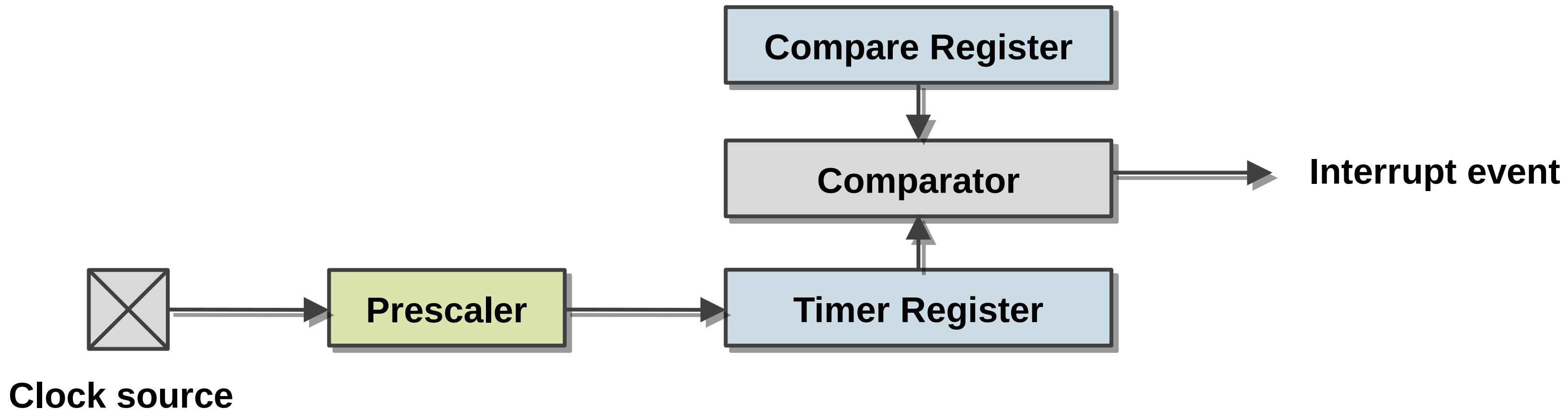
- In contrast, a software timer is a similar function block but implemented in software. Software timers usually
  - Are based on hardware timer;
  - Increase or decrease when interrupted by a hardware timer;
  - Offers a lower level of time precision compared with hardware timer;
  - Can have multiple instances that are more than the actual hardware timers.

Components of a Standard Timer

- A prescaler
  - Takes the clock source as its input
  - Divides the input frequency by a predefined value, e.g. 4, 8, 16…
  - Outputs the divided frequency to the other components;
- A timer register
  - Increases or decreases at a fixed frequency;
  - Driven by the output from the prescaler, often referred as ticks;
- Capture register
  - Loads the current value from the timer register upon the occurrence of certain events;
  - Can also generate an interrupt upon the events;
- Compare register
  - Is loaded with a desired value, which is periodically compared with the value in the timer register;
  - If the two values are the same, an interrupt can be generated.

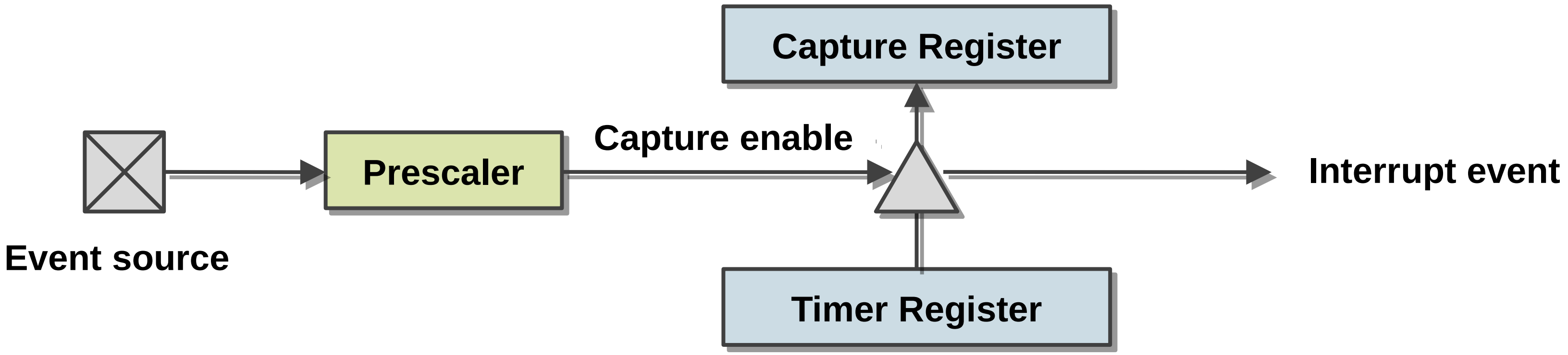
Timer Operation Mode

- Typically, a standard timer may have three operation modes: compare mode, capture mode, and PWM mode.
- Compare mode example
  - Preload the compare register with a desired value;
  - Timer register is incremented or decremented automatically at a frequency from the output of the prescaler;
  - The values in the compare register and the timer register are automatically compared; once equal, an interrupt can be generated and the timer register should be reset.



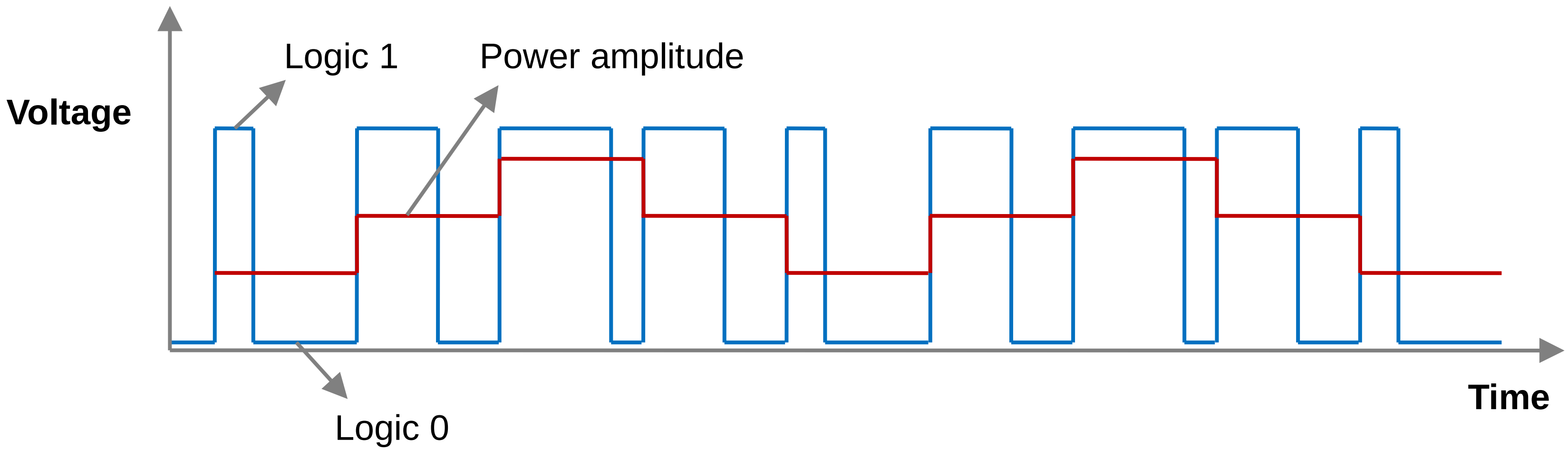
Timer Operation Mode

- Capture mode
  - The event source generates a sequence of pulses;
  - Optionally, the prescaler can be used to divide the frequency of the events;
  - Once the event (or divided events) occurs, the capture will be enabled;
  - The capture register then takes a "snapshot" of the timer register at the moment when the event occurs;
  - Optionally, an interrupt can be generated to notify the processor to do some actions.



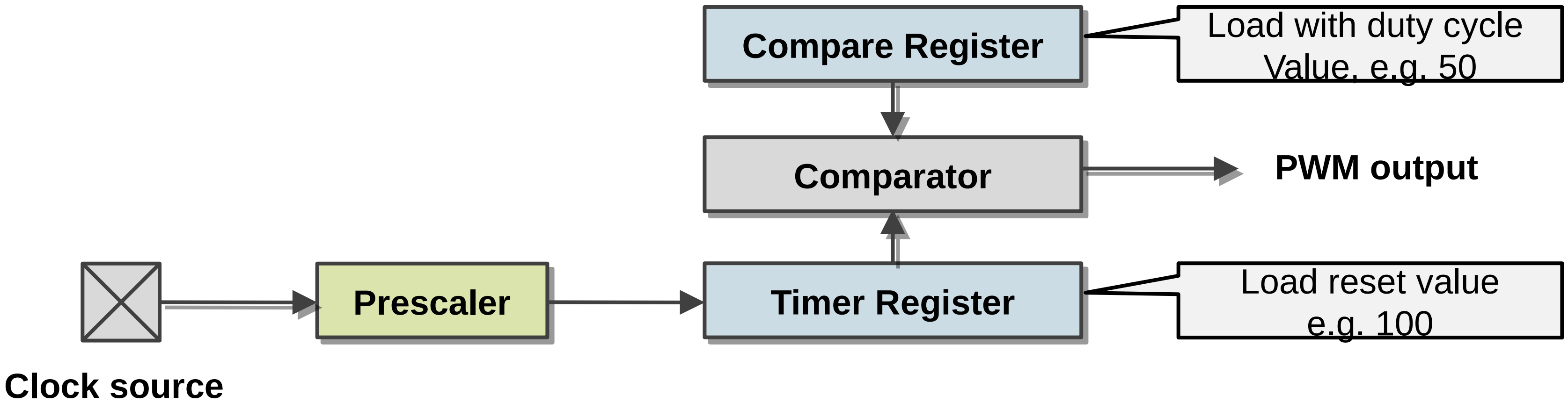
Timer Operation Mode

- Pulse-width modulation (PWM) mode
  - Uses the width of the pulse to modulate an amplitude;
  - The amplitude can be represented by the duty cycle, which describes the proportion of the “1” state in one pulse period;
  - Mainly used for power supplied electrical devices;
  - Pulse frequency ranges from few kHz (e.g. motor drive) to hundreds of kHz (e.g. audio amplifier, computer power supplies);



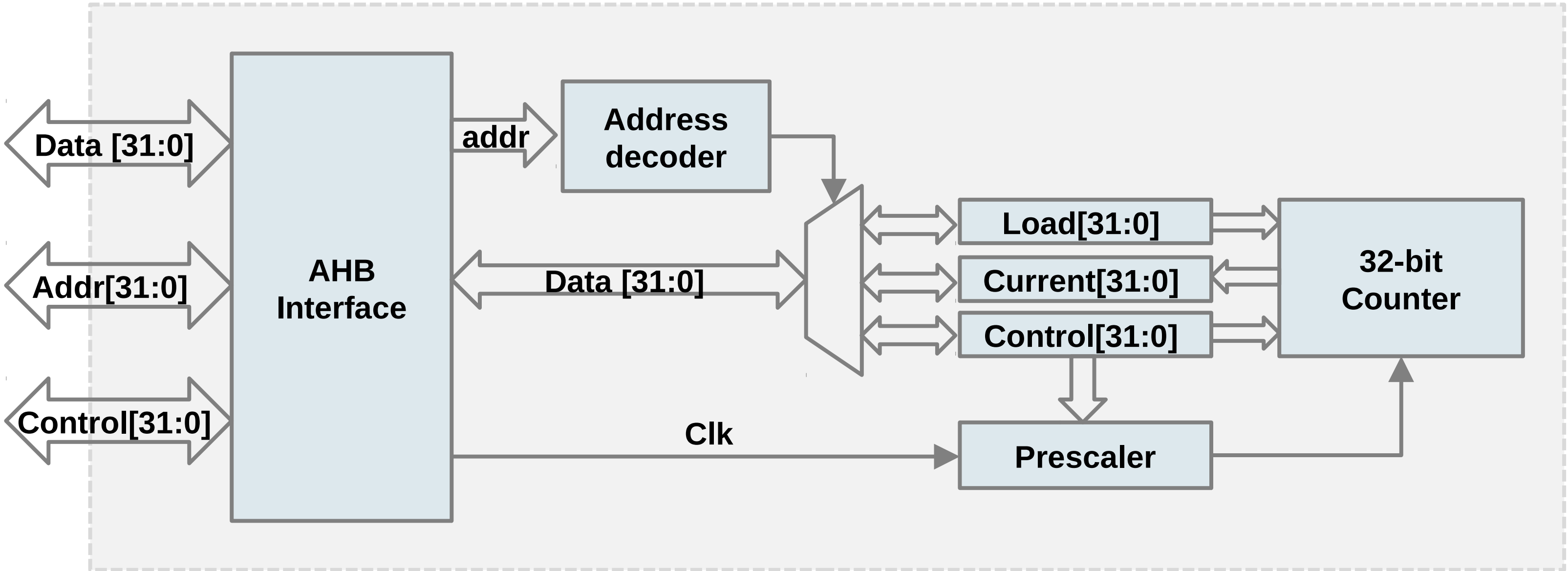
Timer Operation Mode

- Example of PWM mode
  - The PWM mode is similar to the compare mode;
  - For example, to generate a 50% power output:
    - Set timer register to reset when reaching 100;
    - Set compare register to 50;
- In effect, the implementation of the three operation modes can be different between various devices.



AHB Timer

- In this teaching material, we are going to design and implement a simplified timer, which has the following functions:
  - Contains a 32-bits counter that automatically counts downwards once it is enabled;
  - When reaching zero, it is reset to the value in the "load value" register
  - At the same time, an interrupt is generated.



Timer Registers

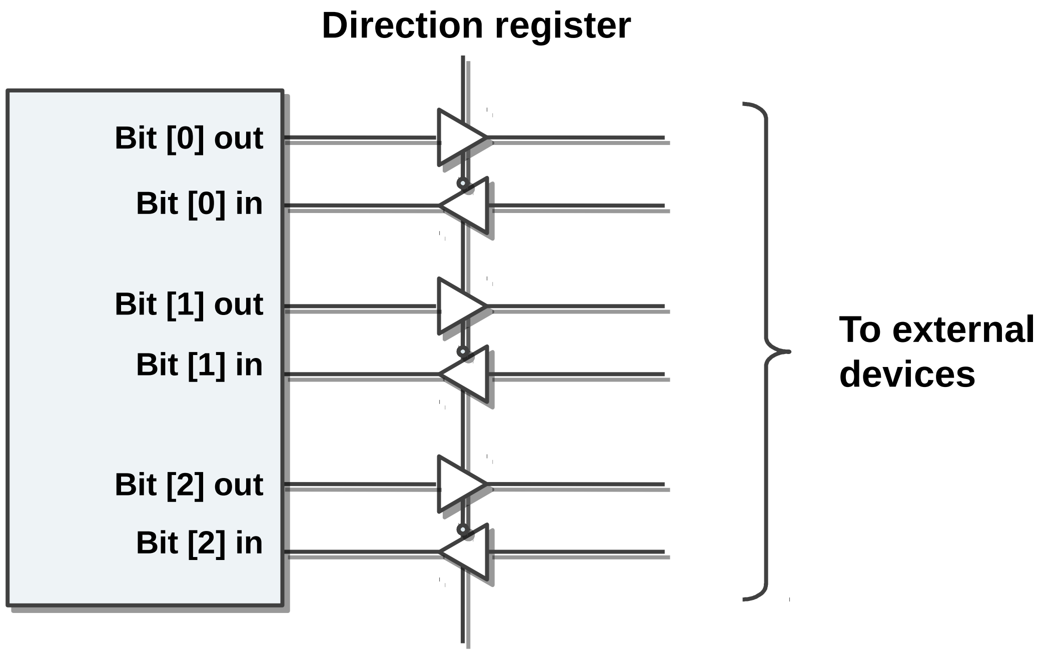
- The timer peripheral should have at least four registers
  - Load value register
    - The reset value when the timer reaches zero;
  - Current value register
    - The current value of the 32-bit counter;
  - Control register
    - Used to start/ stop a counter, and set the prescaler

Register	Address	Size
Base address	0x5300_0000	
Load value	0x5300_0000	4 Byte
Current value	0x5300_0004	4 Byte
Control	0x5300_0008	4 Byte

General Purpose Input/ Output (GPIO)

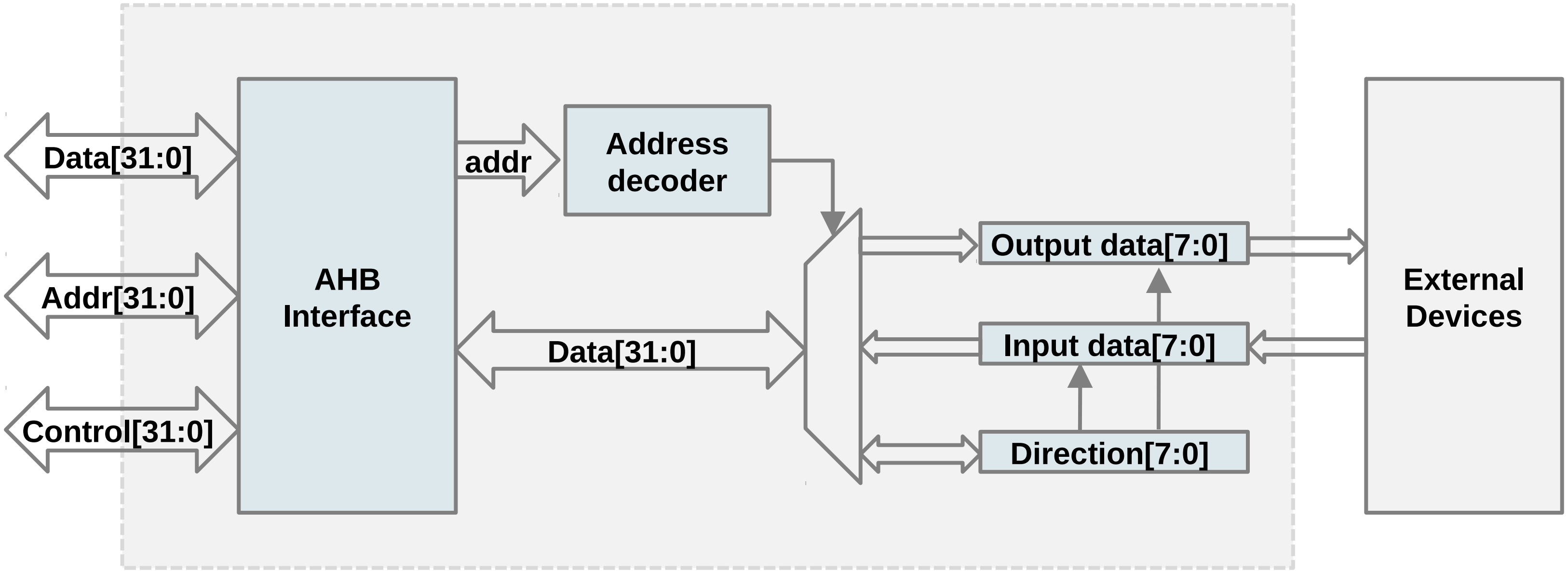
GPIO Overview

- General-purpose input/output (GPIO)
  - Used for general purpose, no special usage defined;
  - Widely used for most of the applications;
  - The direction of input/ output is controlled by the direction register;
  - A mask register is often used to mask out certain bits.



AHB GPIO

- In this set of material, we will design and implement a simple GPIO peripheral
  - Only has the basic registers, namely data in, data out, and direction register;
  - Does not have a mask register or any other functions.



GPIO Registers

- The UART peripheral registers include
  - Data registers
    - Input data – the data read from external devices;
    - Output data – The data sent to external devices;
  - Direction register
    - Controls either it is a read or write operation.

Register	Address	Size
GPIO base address	0x5300_0000	
Data	0x5400_0000	4 Byte
Direction	0x5400_0004	4 Byte

7-SEGMENT DISPLAY

7-Segment Display Overview

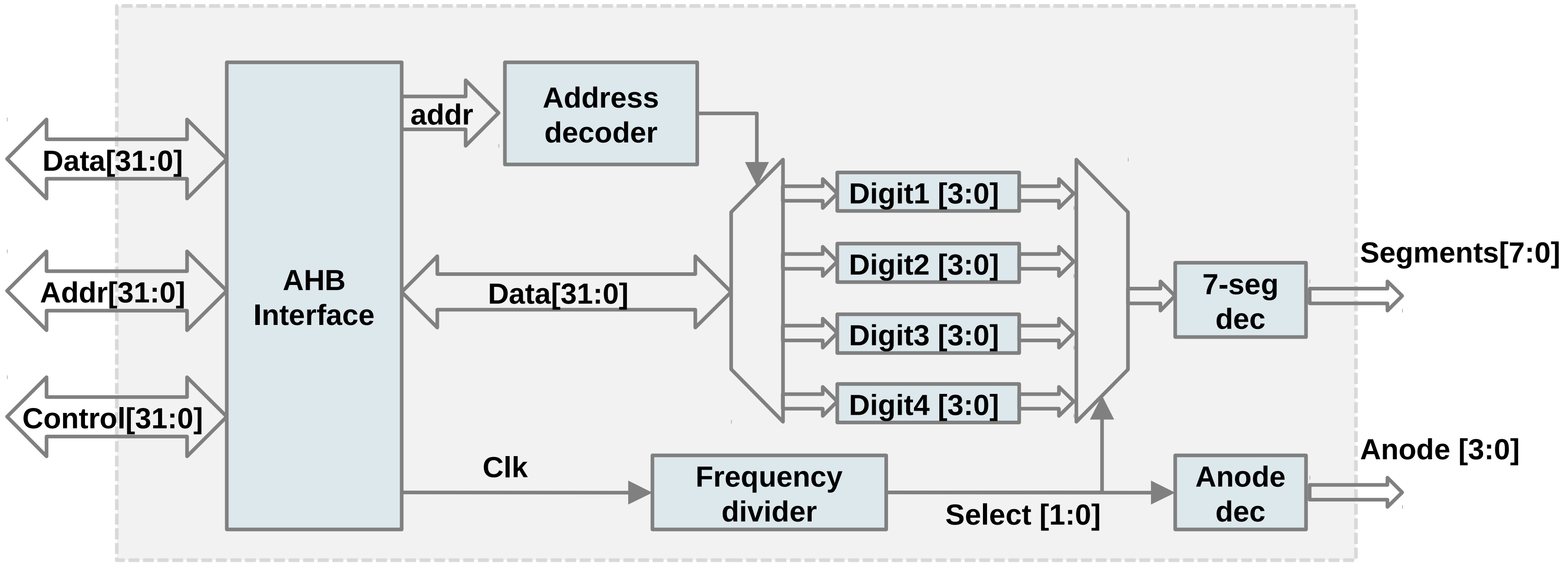
- The 7-segment display uses 7 segments and a dot to display numerals or letters;
- Widely used in digital electronic devices, such as digital clocks, electronic meters;
- Simple control, easy for debugging.

AHB 7-Segment Display

- The implementation of the 7-segment display varies from device to device, for example, Digilent Nexys3 board uses 12 pins to control the 7-segment display:
  - Segment [6:0] -- used to switch on or off one segment;
  - Dot [0:0] -- used to switch the dot bit for one digit;
  - Anode [3:0] -- used to select the four digits, switch on by ‘0’;
- To display different values on four digit, they need to be enabled on one by one. For example, to display “1234”, the sequence can be:
  - Anode[3:0]=0111; segment [6:0] = ‘1’;
  - Anode[3:0]=1011; segment [6:0] = ‘2’ ;
  - …
- The looping frequency can be set to about 1000Hz, which is
  - Slow enough to allow each anode to switch on;
  - Fast enough to give a vision for human eye that all of the digits are on at the same time.

AHB 7-Segment Display

- The values of the four digits are stored in four registers;
- The clock frequency is divided to loop the four digits.



7-Segment Display Registers

- The UART peripheral has four registers
  - Digit1: the first digit on the 7-segment display
  - Digit2: the second digit on the 7-segment display
  - Digit3: the third digit on the 7-segment display
  - Digit4: the forth digit on the 7-segment display

Register	Address	Size
Base address	0x5500_0000	
Digit 1	0x5500_0000	4 Byte
Digit 2	0x5500_0004	4 Byte
Digit 3	0x5500_0008	4 Byte
Digit 4	0x5500_000C	4 Byte

Memory Space

- The memory space for all peripherals is allocated as follow:

Peripheral	Base address	End address	Size
MEM	0x0000_0000	0x4FFF_FFFF	167MB
VGA	0x5000_0000	0x50FF_FFFF	16MB
UART	0x5100_0000	0x51FF_FFFF	16MB
Timer	0x5200_0000	0x52FF_FFFF	16MB
GPIO	0x5300_0000	0x53FF_FFFF	16MB
7-segment	0x5400_0000	0x54FF_FFFF	16MB

Lab Practice

Lab Practice

- Step1- Hardware design
  - Design and implement the peripheral (an AHB timer, a GPIO peripheral, and a 7-segment display) in hardware using Verilog;
- Step2- Software programming
  - Test the peripherals using Cortex-M0 processor programed in assembler language;
- Step3- System demonstration
  - Input data from switches and output them to LEDs;
  - Display the timer value to the 7-segment display.

Useful Resources

- Reference1
  - Nexys3 Reference Manual:

    http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf