



AMBA Bus Architecture

APB Specification

Contents

- 1 Introduction
- 2 Transfers
- 3 Practice
- 4 Exercise

AMBA (1)

- "AMBA (Advanced Microcontroller Bus Architecture) is a freely-available, open standard for the connection and management of functional blocks in a system-on-chip (SoC)." [<https://developer.arm.com/Architectures/AMBA>]
- Real-World Examples of AMBA in Industry
 - ARM Cortex-Based SoCs (e.g., Smartphones, Embedded Systems)
 - ✓ Most ARM Cortex-A/R/M processors internally use AXI, AHB, and APB buses to connect functional blocks.
 - ✓ Examples include Samsung Exynos, Qualcomm Snapdragon, and NXP i.MX series SoCs.
 - FPGA SoCs (e.g., Xilinx Zynq, Intel SoC FPGAs)
 - ✓ Platforms like Xilinx Zynq combine an ARM processing system (PS) with programmable logic (PL), connected via AXI4 and AXI4-Lite buses.
 - ✓ Designers use AMBA-based IP blocks such as AXI Interconnect, AXI DMA, and AXI GPIO in tools like Vivado or Quartus.

AMBA (2)

- Key AMBA Specifications

Spec	Description	AMBA	AMBA 2	AMBA 3	AMBA 4	AMBA 5
CHI	A credited coherency protocol Layered architecture for scalability					CHI
ACE	A superset of AXI – system-wide coherency across multicore clusters				ACE + Lite	ACE5 + Lite
AXI	AXI supports separate A/D phases, bursts, multiple outstanding addresses, and OoO responses			AXI3	AXI4 +Lite, +Stream	AXI5
AHB	AHB supports 64 and 128 bit multi-manager AHB-Lite is for single managers		AHB	AHB + Lite		AHB5 + Lite
APB	System bus for low bandwidth peripherals	APB	APB2	APB3	APB4	

AMBA (3)

■ AMBA

- The earliest AMBA buses included the Advanced System Bus (ASB) and the Advanced Peripheral Bus (APB).
 - ✓ While ASB has been replaced by newer protocols, APB remains in common use today.

■ AMBA 2

- AMBA 2 introduced the AMBA High-performance Bus (AHB).
 - ✓ It operates on a single clock edge.
 - ✓ Each basic transaction includes an address phase followed by a data phase.

■ AMBA 3

- AMBA 3 introduced AHB-Lite as a simplified subset of AHB.
 - ✓ AHB-Lite is optimized for designs with a single bus master.
- The Advanced eXtensible Interface (AXI) was also introduced, targeting high-performance systems operating at high clock frequencies.

AMBA (4)

■ AMBA 4

- AMBA 4 began with AXI4, followed by the AXI Coherency Extensions (ACE).
 - ✓ ACE builds on AXI by adding signaling to support system-wide coherency.
- Additionally, the AXI4-Stream protocol was introduced for unidirectional data transfers from manager to subordinate.

■ AMBA 5

- The AMBA 5 Coherent Hub Interface (CHI) specification was introduced.
 - ✓ It features a redesigned high-speed transport layer for reducing system congestion.
- The AHB-Lite protocol was also updated to AHB5 to support the Armv8-M and extends the TrustZone security foundation from the processor to the broader system.

Introduction (1)

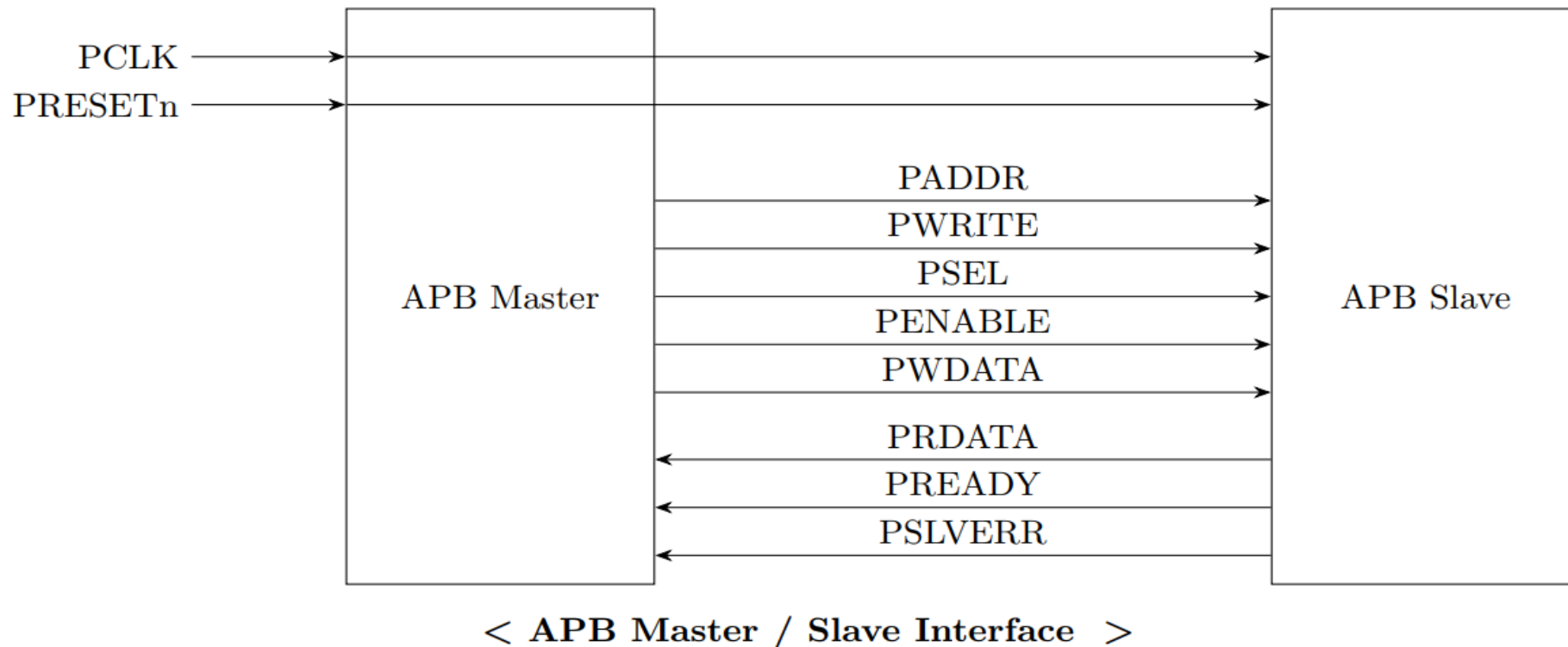
- The APB (Advanced Peripheral Bus) is a member of the AMBA protocol family.
 - ARM IHI 0024C, AMBA APB Protocol v2.0, ARM Ltd.
 - ✓ <https://developer.arm.com/documentation/ihi0024/c/>
 - It offers a simple, low-power, and low-cost interface without pipelining or burst support for low-bandwidth peripherals.
 - Each APB transfer requires a minimum of two clock cycles.
- APB interfaces can be used in conjunction with the following AMBA buses:
 - AHB (Advanced High-performance Bus) / AHB-Lite
 - AXI (Advanced eXtensible Interface) / AXI4-Lite

Introduction (2)

- The APB specification has evolved through several versions:
 - APB2: First introduced in the AMBA 2 specification
 - APB3: Introduced in the AMBA 3 specification (v1.0)
 - ✓ PREADY: Indicates the completion of an APB transfer.
 - ✓ PSLVERR: Signals that a transfer has failed.
 - APB4: Described in the AMBA APB Protocol Specification v2.0
 - ✓ PPROT: Supports secure and non-secure access control.
 - ✓ PSTRB: Enables sparse write access to the data bus using strobe signals.

Signal Descriptions (1)

- This diagram presents a basic APB structure with one master / slave.
 - It does not include APB4 signals such as PSTRB, PPROT and additional components like bridges and decoders.
 - The focus here is on understanding the basic APB3 transfer mechanism.



Signal Descriptions (2)

- The descriptions of the basic APB signals.

Signal	Source	Meaning	Description
PCLK	Clock source	Clock	All transfers on the APB are synchronized to the rising edge of PCLK.
PRESETn	External device	Reset	The APB reset signal is active LOW and is usually tied directly to the system bus reset line.
PADDR	APB Master	Address	It initiates address signals during data transfers. It can be up to 32 bits wide.
PSEL	APB Master	Select	It indicates that the slave device is selected and that a data transfer is required.
PENABLE	APB Master	Enable	It marks the enable phase, starting from the second cycle of a data transfer.
PWRITE	APB Master	Direction	It defines the direction of a data transfer - write when HIGH, read when LOW.
PWDATA	APB Master	Write Data	It carries the write data from the master to the slave during write operations. It can be up to 32 bits wide.
PREADY	APB Slave	Ready	It is used by the slave to extend the duration of a data transfer.
PRDATA	APB Slave	Read Data	It carries the read data from the slave to the master during read operations. It can be up to 32 bits wide.
PSLVERR	APB Slave	Failure	It signals an error from the slave to the master during a data transfer.

Contents

- 1 Introduction
- 2 Transfers
- 3 Practice
- 4 Exercise

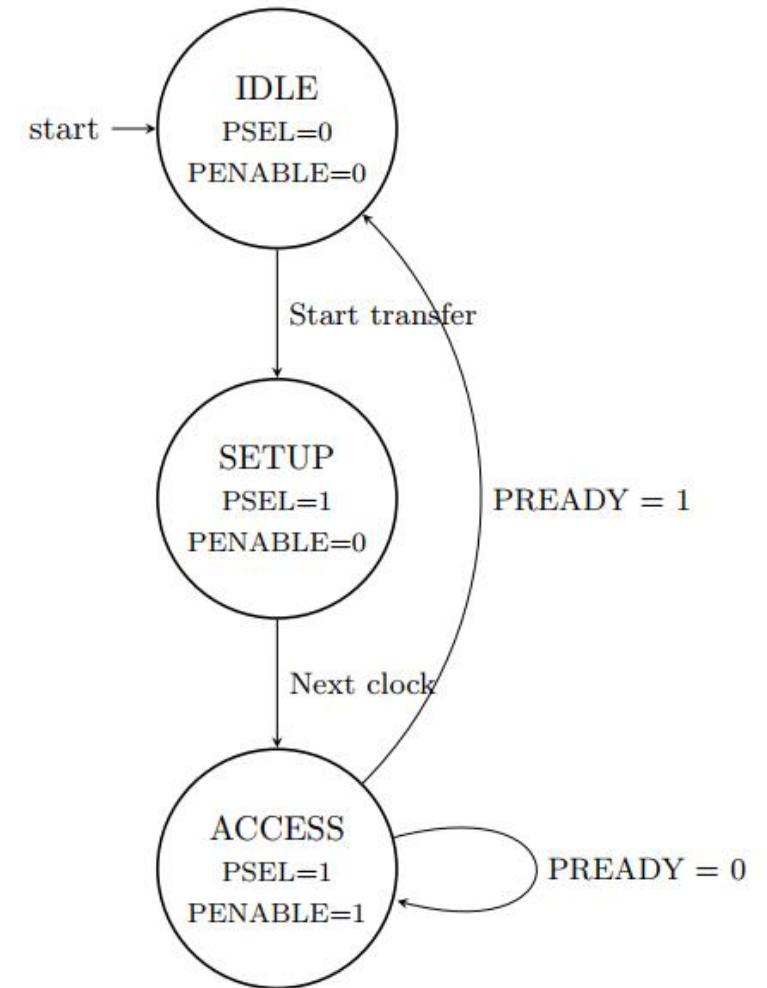
Operating States (1)

■ IDLE

- The bus remains inactive during this state.
- No valid transfer is in progress.
 - ✓ PSEL = 0 and PENABLE = 0

■ SETUP

- The bus enters the SETUP when a transfer begins.
 - ✓ Asserting PSEL = 1 initiates the transfer.
- The control signals are held stable during this state.
 - ✓ PADDR holds the address of the target.
 - ✓ PWRITE defines the direction of the transfer.
 - ✓ (1 = write, 0 = read).
 - ✓ PWDATA holds valid write data during a write operation.
- This state lasts one clock cycle.
 - ✓ The bus moves to the ACCESS on the next cycle.

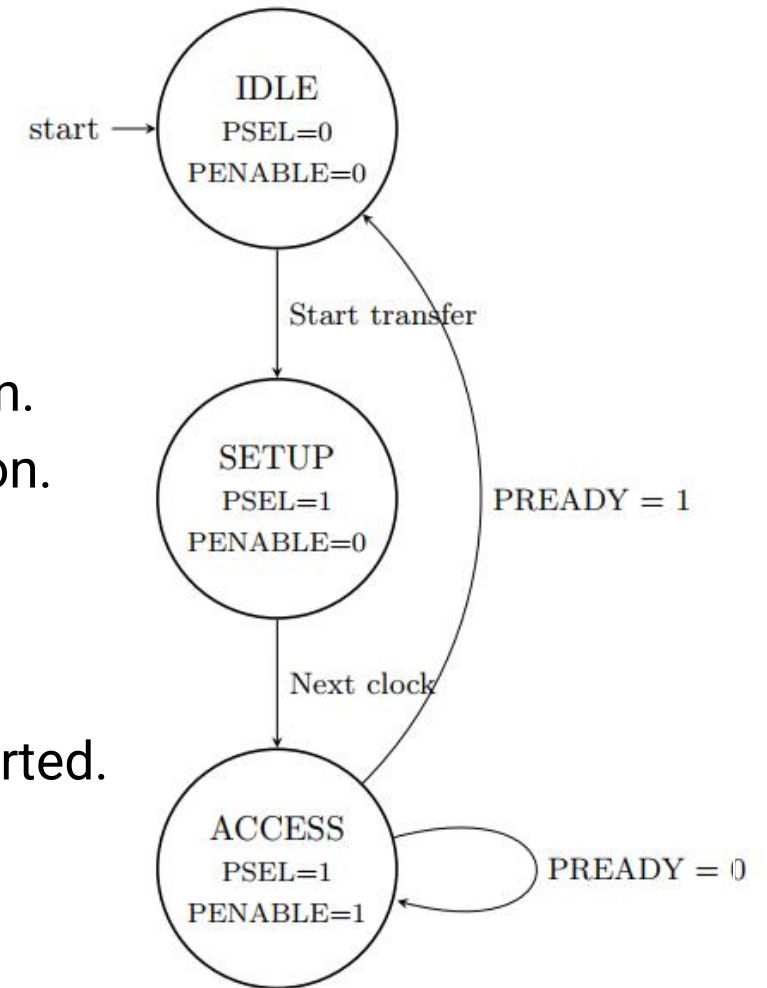


< APB Operating States >

Operating States (2)

■ ACCESS

- This is the state where the actual data transfer occurs.
 - ✓ PSEL = 1 and PENABLE = 1
- All control signals must remain stable.
 - ✓ PADDR, PWRITE, PWDATA
- The slave captures data from PWDATA for a write operation.
- The master captures data from PRDATA for a read operation.
- PREADY indicates whether the slave is ready
 - ✓ If PREADY = 1, the transfer completes in this cycle.
 - ✓ If PREADY = 0, the master waits in this state until ready.
- After a successful transfer, PSEL and PENABLE are deasserted.
 - ✓ The bus returns to the IDLE.

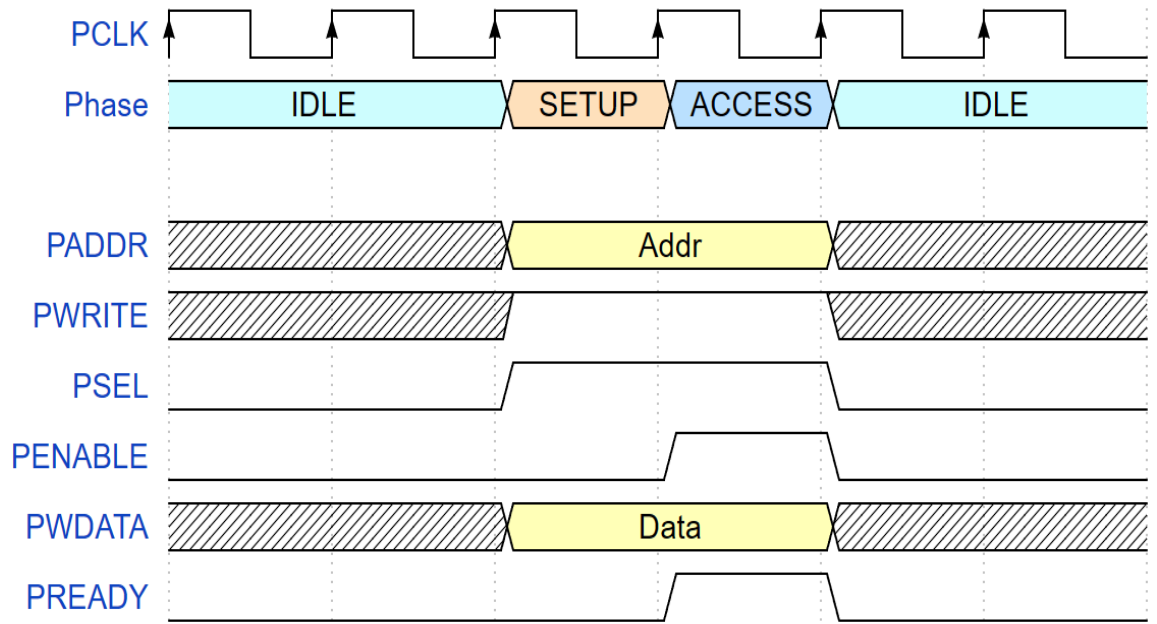


< APB Operating States >

Write Transfers (1)

■ Write transfer without Wait

- Cycle 1~2 (IDLE)
 - ✓ PSEL = 0, PENABLE = 0
- Cycle 3 (SETUP)
 - ✓ PSEL = 1, PWRITE = 1
 - ✓ Valid PADDR and PWDATA
 - ✓ Write transfer begins.
- Cycle 4 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 1
 - ✓ The transfer completes.
- Cycle 5 (Back to IDLE)
 - ✓ PSEL = 0, PENABLE = 0

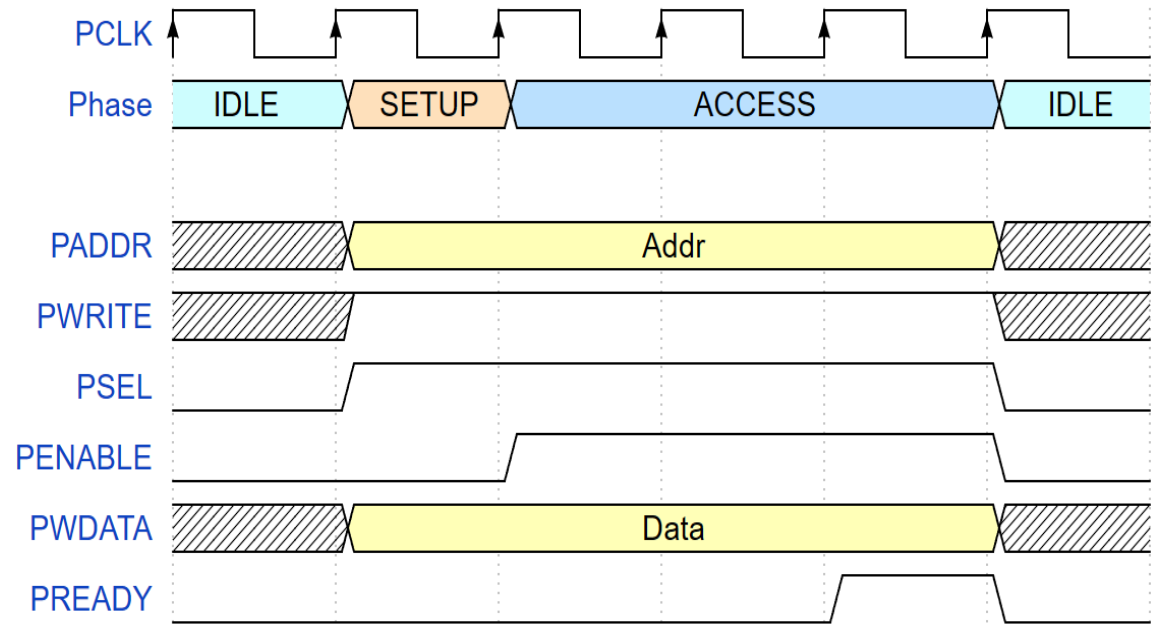


< APB Write Transaction without Wait >

Write Transfers (2)

■ Write transfer with Wait

- Cycle 1~2 (IDLE)
 - ✓ PSEL = 0, PENABLE = 0
- Cycle 3 (SETUP)
 - ✓ PSEL = 1, PWRITE = 1
 - ✓ Valid PADDR and PWDATA
 - ✓ Write transfer begins.
- Cycle 4~5 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 0
 - ✓ Transfer is not yet completed.
- Cycle 6 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 1
 - ✓ The transfer completes.
- Cycle 7 (Back to IDLE)
 - ✓ PSEL = 0, PENABLE = 0

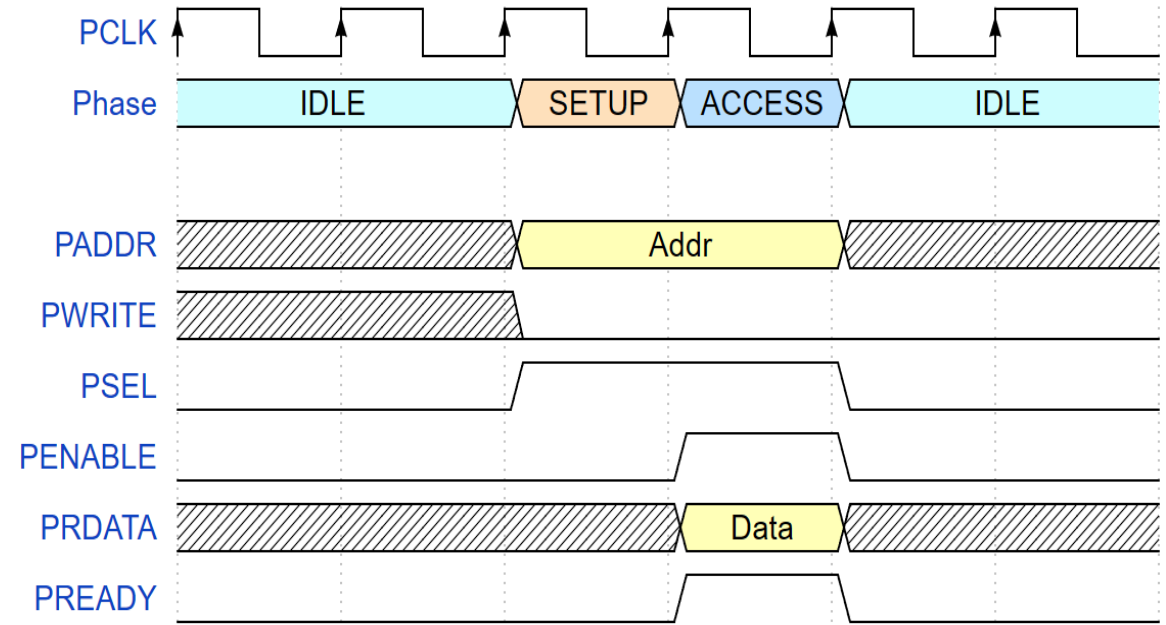


< APB Write Transaction with Wait >

Read Transfers (1)

■ Read transfer without Wait

- Cycle 1~2 (IDLE)
 - ✓ PSEL = 0, PENABLE = 0
- Cycle 3 (SETUP)
 - ✓ PSEL = 1, PWRITE = 0
 - ✓ Valid PADDR
 - ✓ Read transfer begins.
- Cycle 4 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 1
 - ✓ The transfer completes.
- Cycle 5 (Back to IDLE)
 - ✓ PSEL = 0, PENABLE = 0

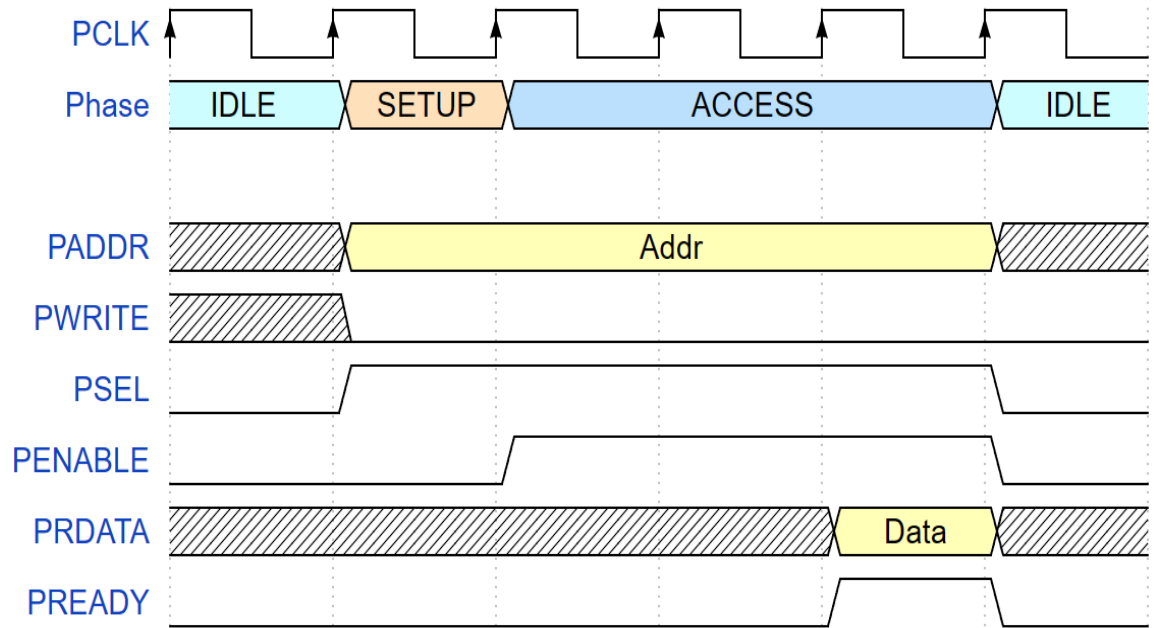


< APB Read Transaction without Wait >

Read Transfers (2)

■ Read transfer with Wait

- Cycle 1~2 (IDLE)
 - ✓ PSEL = 0, PENABLE = 0
- Cycle 3 (SETUP)
 - ✓ PSEL = 1, PWRITE = 0
 - ✓ Valid PADDR
 - ✓ Data transfer begins.
- Cycle 4~5 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 0
 - ✓ Transfer is not yet completed.
- Cycle 6 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 1
 - ✓ The transfer completes.
- Cycle 7 (Back to IDLE)
 - ✓ PSEL = 0, PENABLE = 0



< APB Read Transaction with Wait >

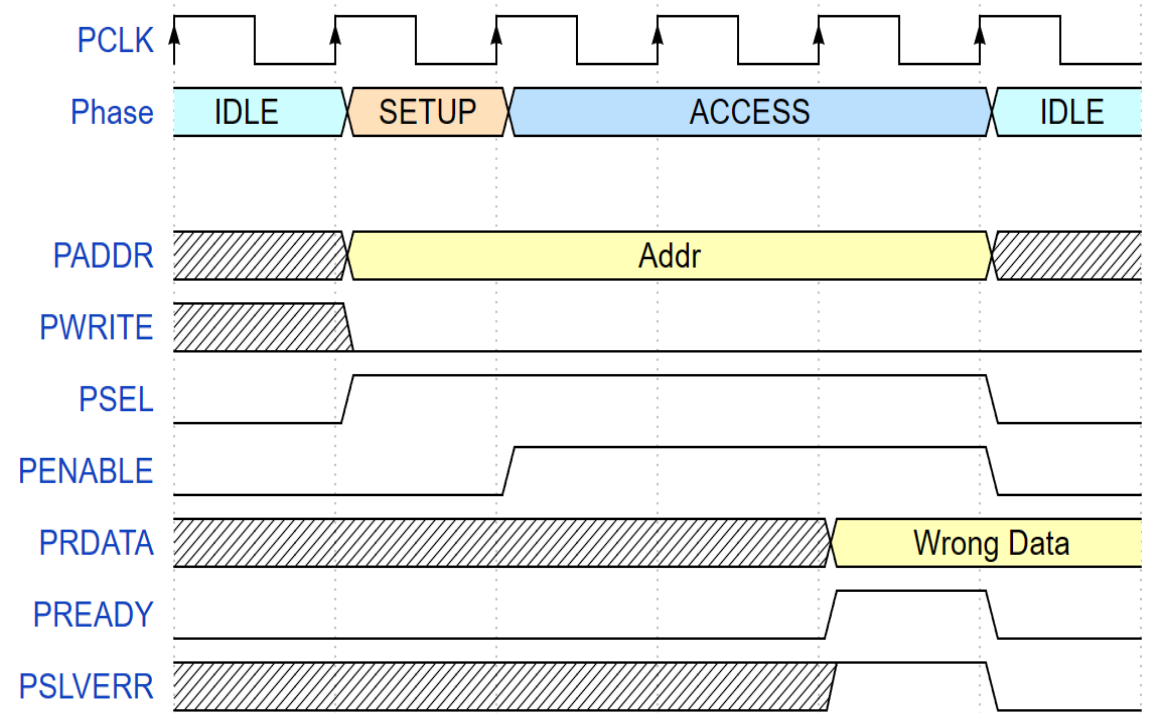
Error Response (1)

- PSLVERR can be used to signal an error during an APB transfer.
- The signal is only valid in the final cycle of the transfer.
 - When PSEL, PENABLE, and PREADY are all HIGH.
- It is recommended to keep PSLVERR LOW whenever any of PSEL, PENABLE, or PREADY is LOW.
- In the case of a transfer that results in an error, the peripheral's state may or may not have been altered
 - An error during a write transfer does not necessarily mean that the register was left unchanged.
 - An error during a read transfer may result in invalid data being returned.

Error Response (2)

■ Read transfer with Error

- Cycle 1~2 (IDLE)
 - ✓ PSEL = 0, PENABLE = 0
- Cycle 3 (SETUP)
 - ✓ PSEL = 1, PWRITE = 0
 - ✓ Valid PADDR
 - ✓ Data transfer begins.
- Cycle 4~5 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 0
 - ✓ Transfer is not yet completed.
- Cycle 6 (ACCESS)
 - ✓ PENABLE = 1, PREADY = 1, PSLVERR = 1
 - ✓ The transfer fails.
- Cycle 7 (Back to IDLE)
 - ✓ PSEL = 0, PENABLE = 0



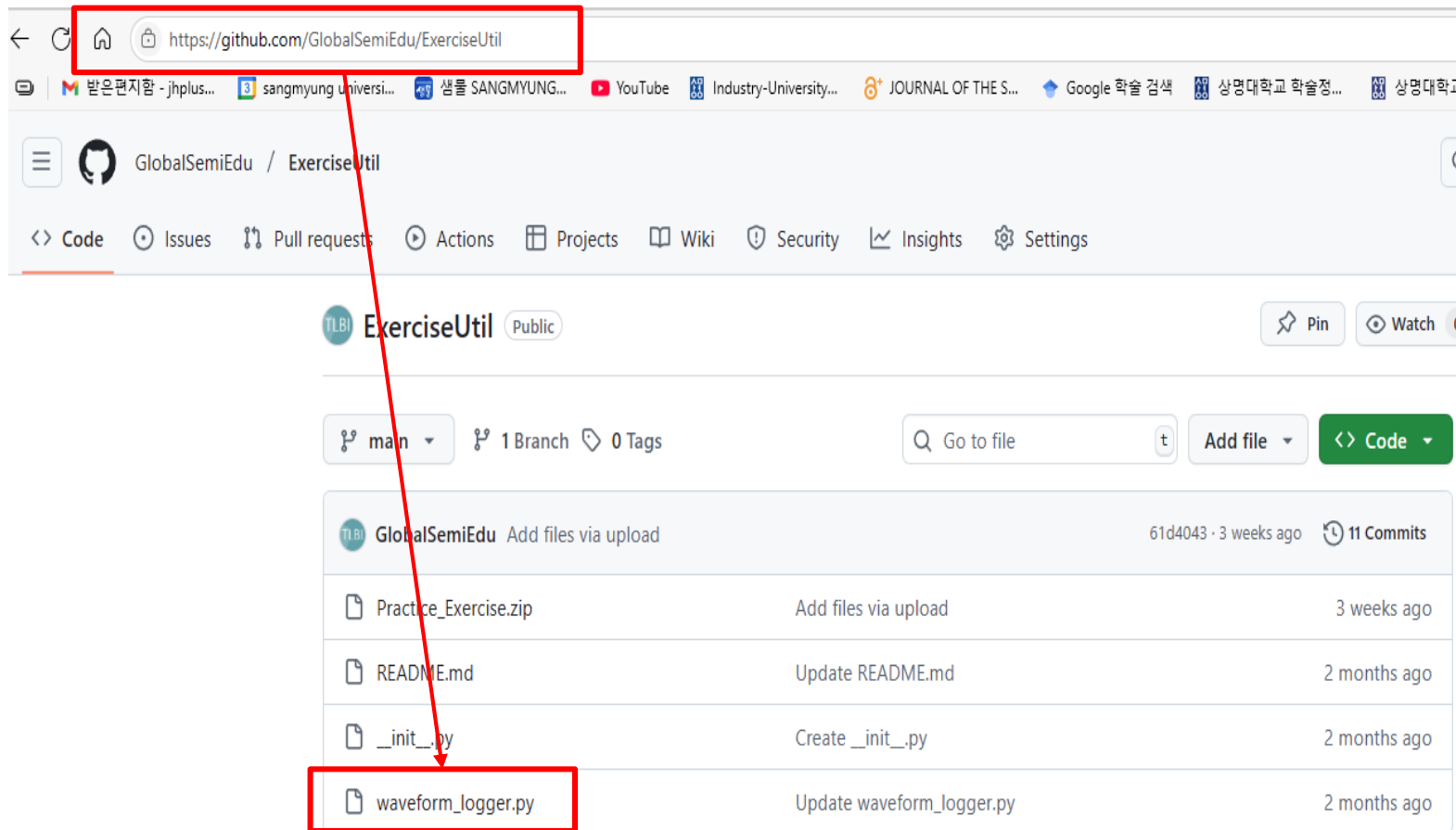
< APB Read Transaction with Error Response >

Contents

- 1 Introduction
- 2 Transfers
- 3 Practice
- 4 Exercise

Environment Setup (1)

- 1) FPGA와 연결된 PC에 다음의 파일을 다운로드 할것.
 - https://github.com/GlobalSemiEdu/ExerciseUtil/waveform_logger.py



Environment Setup (2)

- https://github.com/GlobalSemiEdu/ExerciseUtil/waveform_logger.py

ExerciseUtil / waveform_logger.py

GlobalSemiEdu Update waveform_logger.py

Code

Blame

50 lines (42 loc) · 1.63 KB

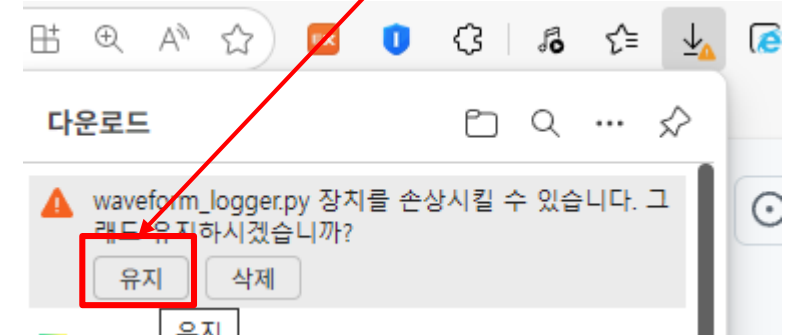


Download raw file

Raw

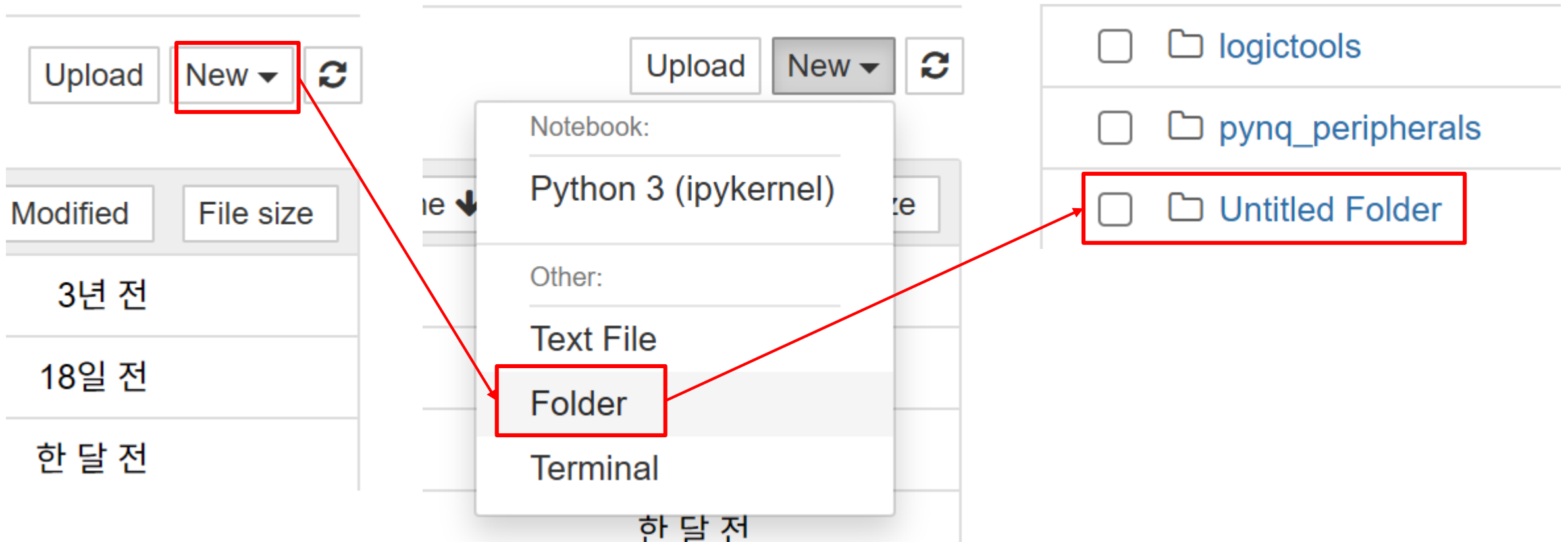


```
1  class WaveformLogger:
2  def __init__(self):
3      self.log = {}
4      self.prev = {}
5      self.clk_sig = None
6      self.ctrl_sigs = None
7      self.data_sigs = None
8
```



Environment Setup (3)

- 2) Jupyter 환경에서 다음의 새로운 폴더를 생성
 - Step 1: Create a new folder



Environment Setup (4)

- Step 2: waveform_logger.py 파일을 업로드

Select items to perform actions on them.

☐ 0

▼

📁 / Untitled Folder

Name ▼

Last Modified

File size

📁 ..

몇 초 전

The notebook list is empty.

Upload

New ▼

↺

waveform_logger.py

Files

Running

Clusters

Nbextensions

Select items to perform actions on them.

☐ 0

▼

📁 / Untitled Folder

Name ▼

Last Modified

File size

📄 waveform_logger.py

Upload

Cancel

📁 ..

몇 초 전

Environment Setup (5)

- Step 3: 새로운 notebook을 생성



Practice 1 (2) - revisited

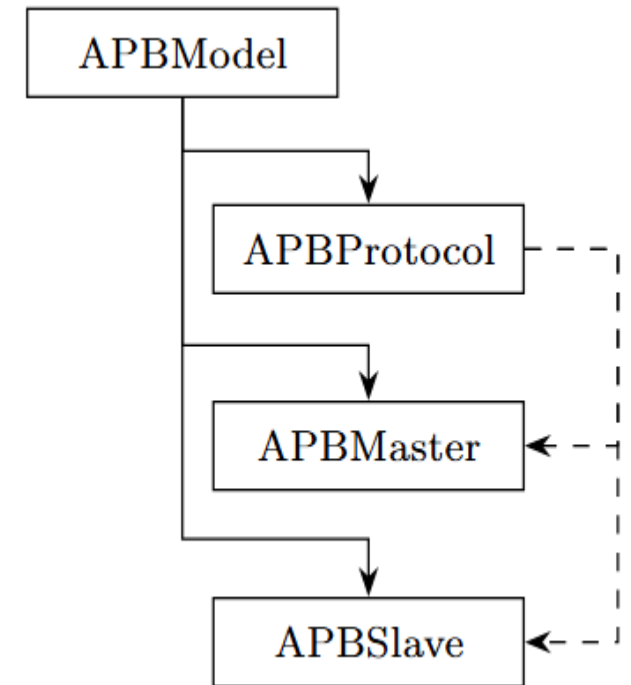
- Setup

```
!git clone https://github.com/GlobalSemiEdu/ExerciseUtil.git
```

```
from ExerciseUtil.waveform_logger import WaveformLogger  
from waveform_logger import WaveformLogger  
from pynq.lib.logictools.waveform import draw_wavedrom
```

Practice 1 (1)

- This practice uses Python to simulate the behavior of the APB protocol.
- Components
 - APBProtocol class
 - ✓ It initializes and holds all shared APB signals.
 - APBMaster class
 - ✓ It generates transfers (WRITE or READ) using a simple FSM.
 - APBSlave class
 - ✓ It responds to master signals.
 - ✓ It implements memory for data.
 - APBModel class
 - ✓ It coordinates the master, slave, and clock ticks.
 - ✓ The method tx_start() initiates a new APB transaction.
 - ✓ The method step() advances the simulation by one clock cycle.



Practice 1 (2)

- Setup

```
!git clone https://github.com/GlobalSemiEdu/ExerciseUtil.git
```

```
from ExerciseUtil.waveform_logger import WaveformLogger  
from pynq.lib.logictools.waveform import draw_wavedrom
```

```
Cloning into 'ExerciseUtil'...  
remote: Enumerating objects: 18, done.  
remote: Counting objects: 100% (18/18), done.  
remote: Compressing objects: 100% (14/14), done.  
remote: Total 18 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)  
Receiving objects: 100% (18/18), 6.96 KiB | 593.00 KiB/s, done.  
Resolving deltas: 100% (2/2), done.
```

Practice 1 (3)

■ APBProtocol

```
class APBProtocol:  
→ CtrlSigs = ["PSEL", "PENABLE", "PWRITE", "PREADY"]  
→ DataSigs = ["PWDATA", "PRDATA", "PADDR"]  
→ ClkSig = "PCLK"  
→ AllSigs = [ClkSig] + CtrlSigs + DataSigs  
  
→ def __init__(self):  
→→ self.signals = {}  
→→ for sig in self.AllSigs:  
→→→ self.signals[sig] = 0
```

Practice 1 (4)

■ APBMaster

```
class APBMaster:
→def __init__(self, signals):
→→self.signals = signals
→→self.state = "IDLE"

→def write(self, addr, data):
→→self.state = "SETUP"
→→self.signals["PADDR"] = addr
→→self.signals["PWRITE"] = 1
→→self.signals["PWDATA"] = data

→def read(self, addr):
→→self.state = "SETUP"
→→self.signals["PADDR"] = addr
→→self.signals["PWRITE"] = 0
```

```
→def tick(self):
→→sig = self.signals
→→sig["PSEL"] = 0
→→sig["PENABLE"] = 0

→→if self.state == "SETUP":
→→→sig["PSEL"] = 1
→→→sig["PENABLE"] = 0
→→→self.state = "ACCESS"
→→elif self.state == "ACCESS":
→→→if sig["PREADY"] == 1:
→→→→self.state = "IDLE"
→→else:
→→→sig["PSEL"] = 1
→→→sig["PENABLE"] = 1
```

Practice 1 (5)

■ APBSlave

```
class APBSlave:
→def __init__(self, signals):
→→self.signals = signals
→→self.memory = {}

→def tick(self):
→→sig = self.signals
→→self.signals["PREADY"] = 0
→→if sig["PSEL"] == 1:
→→→if sig["PENABLE"] == 0:
→→→→self.addr = sig["PADDR"]
→→→→else:
→→→→self.signals["PREADY"] = 1
→→→→if sig["PWRITE"] == 1:
→→→→→self.memory[self.addr] = sig["PWDATA"]
→→→→→else:
→→→→→sig["PRDATA"] = self.memory[self.addr]
```

Practice 1 (6)

■ APBModel

```
class APBModel:
→def __init__(self, logger):
→→self.protocol = APBProtocol()
→→self.logger = logger
→→signals = self.protocol.signals
→→self.logger.init_signals(self.protocol)
→→self.master = APBMaster(signals)
→→self.slave = APBSlave(signals)
→→self.cycle = 0

→def read(self, addr):
→→self.master.read(addr)

→def write(self, addr, data):
→→self.master.write(addr, data)
```

```
→def step(self):
→→self.master.tick()
→→self.slave.tick()
→→self.logger.record(self.protocol.signals)
→→self.cycle += 1

→def is_done(self):
→→return self.master.state == "IDLE"
```


Practice 1 (7)

■ Main code

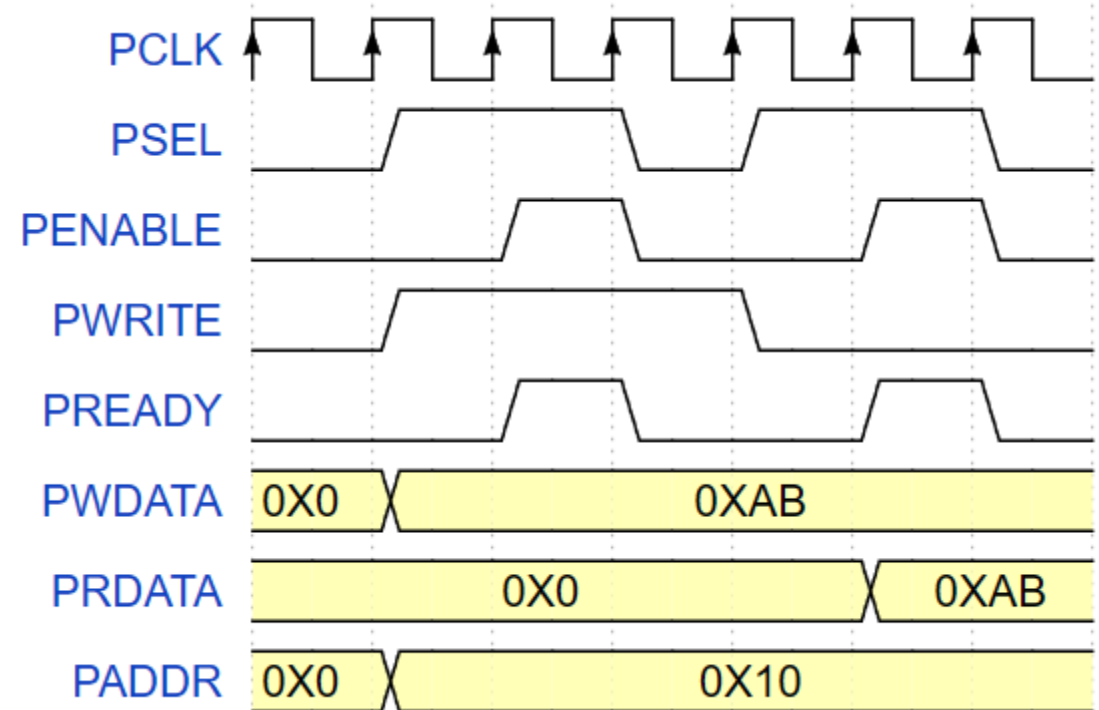
```
logger = WaveformLogger()
apb = APBModel(logger)

apb.write(0x10, 0xAB)
while True:
    →if apb.is_done():
    →→break
    →apb.step()

apb.read(0x10)
while True:
    →if apb.is_done():
    →→break
    →apb.step()

draw_wavedrom(logger.to_json())
```

■ Waveform



Practice 2 (1)

- When a Slave Error occurs:
 - For write operations, it is recommended that the slave does not update memory.
 - For read operations, the slave may return invalid or default data.
 - The behavior after an error is peripheral-specific.
- This practice modifies the APBProtocol and APBSlave to support PSLVERR.
 - The added parts are highlighted in red.
 - Unchanged parts may be partially omitted for clarity.
- Practice scenario
 - The APB Master initiates a write transfer with no error.
 - The APB Master initiates a read transfer that results in an error.
 - ✓ Invalid data (0xDEAD) is returned.
 - A second read transfer is performed without error.
 - ✓ Correct data is returned from memory.

Practice 2 (2)

- APBProtocol - Revisited for PSLVERR

```
class APBProtocol:
→ CtrlSigs = ["PSEL", "PENABLE", "PWRITE", "PREADY", "PSLVERR"]
→ DataSigs = ["PWDATA", "PRDATA", "PADDR"]
→ ClkSig = "PCLK"
→ AllSigs = [ClkSig] + CtrlSigs + DataSigs

→ def __init__(self):
→→ self.signals = {}
→→ for sig in self.AllSigs:
→→→ self.signals[sig] = 0
```

Practice 2 (3)

- APBSlave - Revisited for PSLVERR

```
class APBSlave:
    ...
    → def tick(self):
    → → sig = self.signals
    → → self.signals["PREADY"] = 1
    → → self.signals["PSLVERR"] = 0
    → → if sig["PSEL"] == 1:
    ...
    → → → if sig["PWRITE"] == 1:
    → → → → self.memory[self.addr] = sig["PWDATA"]
    → → → → else:
    → → → → sig["PRDATA"] = self.memory[self.addr]
    → → → → if self.addr in self.memory:
    → → → → → sig["PRDATA"] = self.memory[self.addr]
    → → → → → else:
    → → → → → sig["PRDATA"] = 0xDEAD
    → → → → → self.signals["PSLVERR"] = 1
```

Practice 2 (4)

■ Main code

```
logger = WaveformLogger()
apb = APBModel(logger)
```

```
apb.write(0x10, 0xAB)
while True:
    →if apb.is_done():
    →→break
    →apb.step()
```

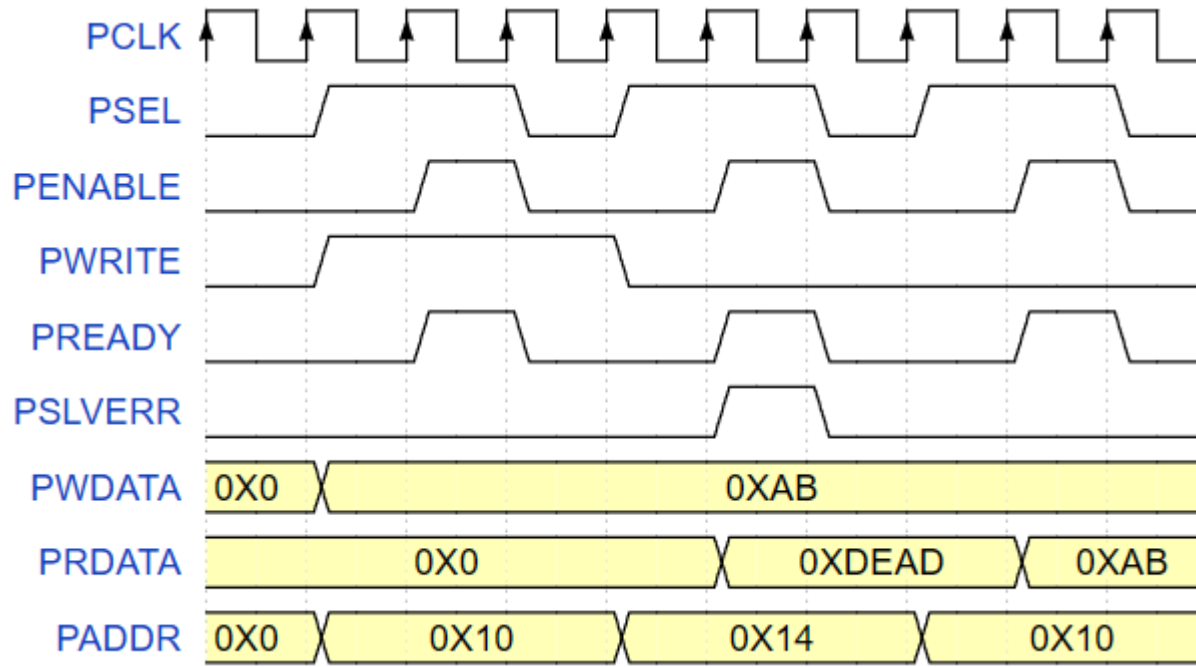
```
apb.read(0x14)
while True:
    →if apb.is_done():
    →→break
    →apb.step()
```

```
apb.read(0x10)
while True:
    →if apb.is_done():
    →→break
    →apb.step()
```

```
draw_wavedrom(logger.to_json())
```

Practice 2 (5)

- Waveform



APB RTL

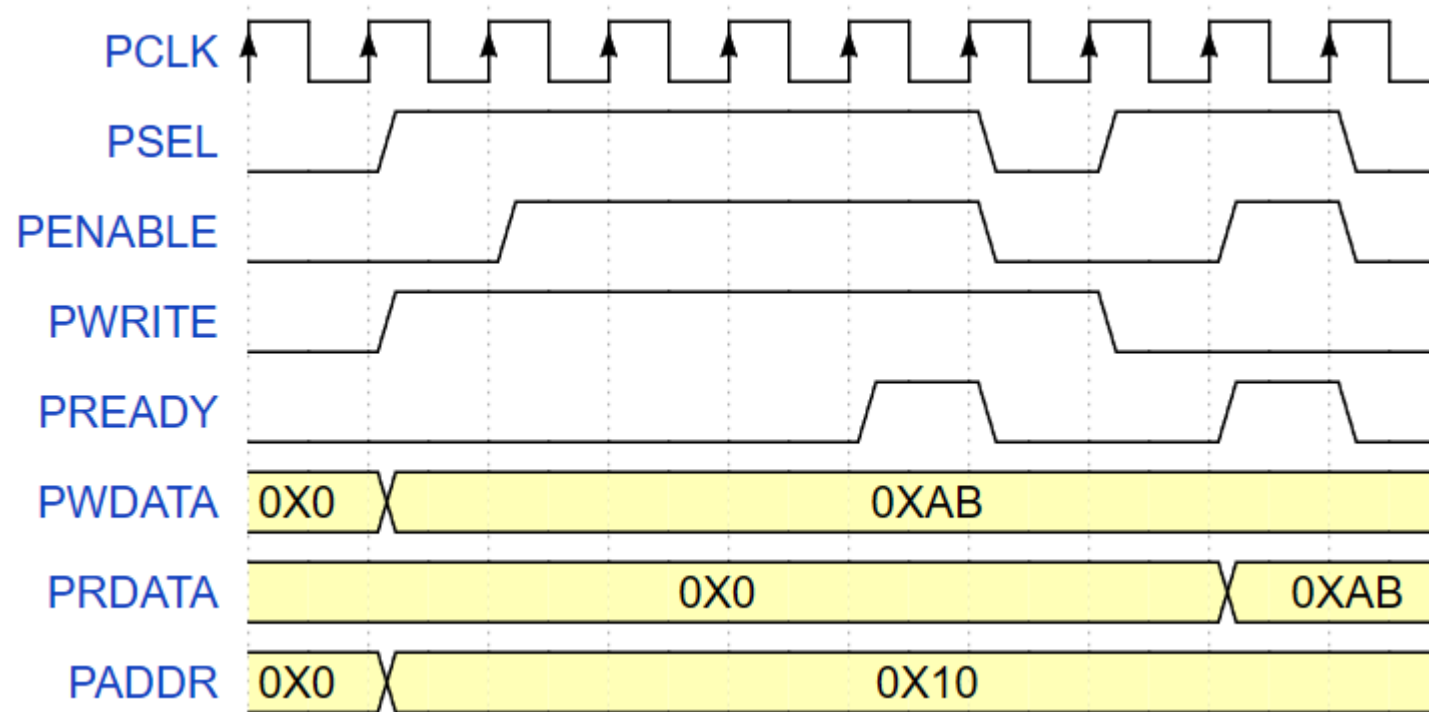
- [iammituraj/apb](https://github.com/iammituraj/apb): APB master and slave developed in RTL.
- <https://github.com/iammituraj/apb>

Contents

- 1 Introduction
- 2 Transfers
- 3 Practice
- 4 Exercise

Exercise (1)

- Modify APBSlave to add a 3-cycle wait state for all read/write transfers.
 - PSLVERR handling is not included in this model.
- Waveform



Exercise (2)

- Main code

```
logger = WaveformLogger()
apb = APBModel(logger)

apb.write(0x10, 0xAB)
while True:
    →if apb.is_done():
    →→break
    →apb.step()

apb.read(0x10)
while True:
    →if apb.is_done():
    →→break
    →apb.step()

draw_wavedrom(logger.to_json())
```

Exercise (3)

■ APBSlave

```
class APBSlave:
→def __init__(self, signals):
→→self.signals = signals
→→self.memory = {}
→→self.wait_count = 3

→def tick(self):
...
→→→else:
→→→→if self.wait_count > 0:
→→→→→self.wait_count -= 1
→→→→else:
→→→→→self.signals["PREADY"] = 1
→→→→→if sig["PWRITE"] == 1:
→→→→→→self.memory[self.addr] = sig["PWDATA"]
...
```

Setup Q/A

- Jupyter Notebook이 안될 때


- 1) UART Console창으로 FPGA에 연결
- 2) 다음의 명령어를 수행

```
xilinx@pynq:~$ sudo vim /etc/network/interfaces.d/eth0  
[sudo] password for xilinx: <- 여기서 암호 입력 : xilinx
```

- 3) 다음의 내용으로 수정하고 저장

```
auto eth0  
iface eth0 inet dhcp
```

```
auto eth0:1  
iface eth0:1 inet static  
—address 192.168.2.99  
—netmask 255.255.255.0
```



```
auto eth0  
iface eth0 inet static  
address 192.168.2.99  
netmask 255.255.255.0
```

- 4) 시스템을 다시 부팅

```
xilinx@pynq:~$ sudo reboot
```