AMBA APB – Case Study

Testing & Verification

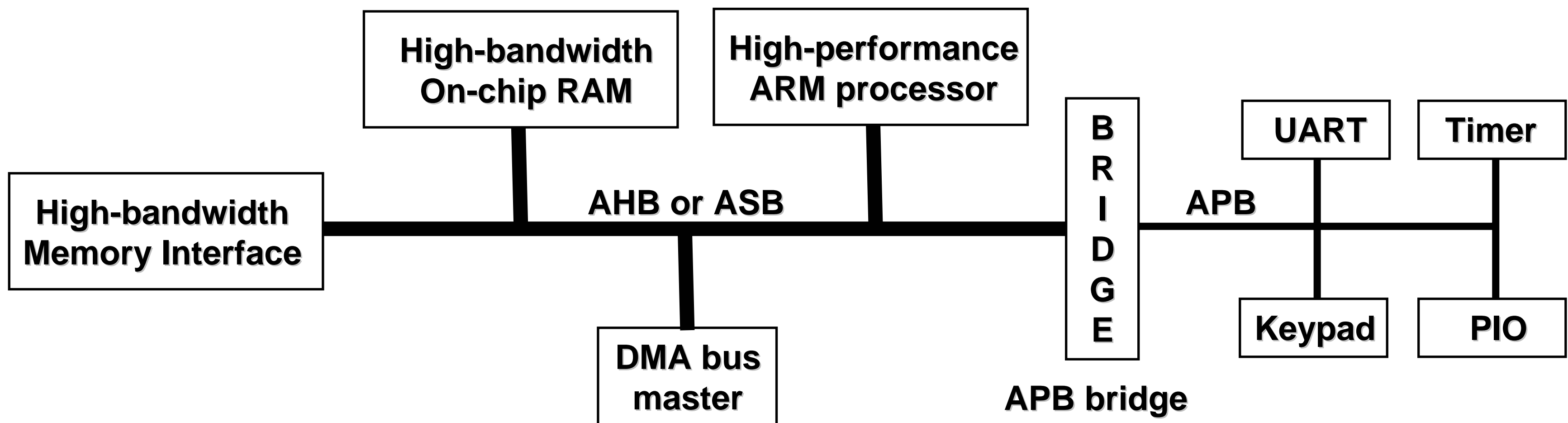
Dept. of Computer Science & Engg, IIT Kharagpur

Pallab Dasgupta

Professor, Dept. of Computer Science & Engg.,
Professor-in-charge, AVLSI Design Lab,
Indian Institute of Technology Kharagpur

Advanced Microcontroller Bus Architecture (AMBA)

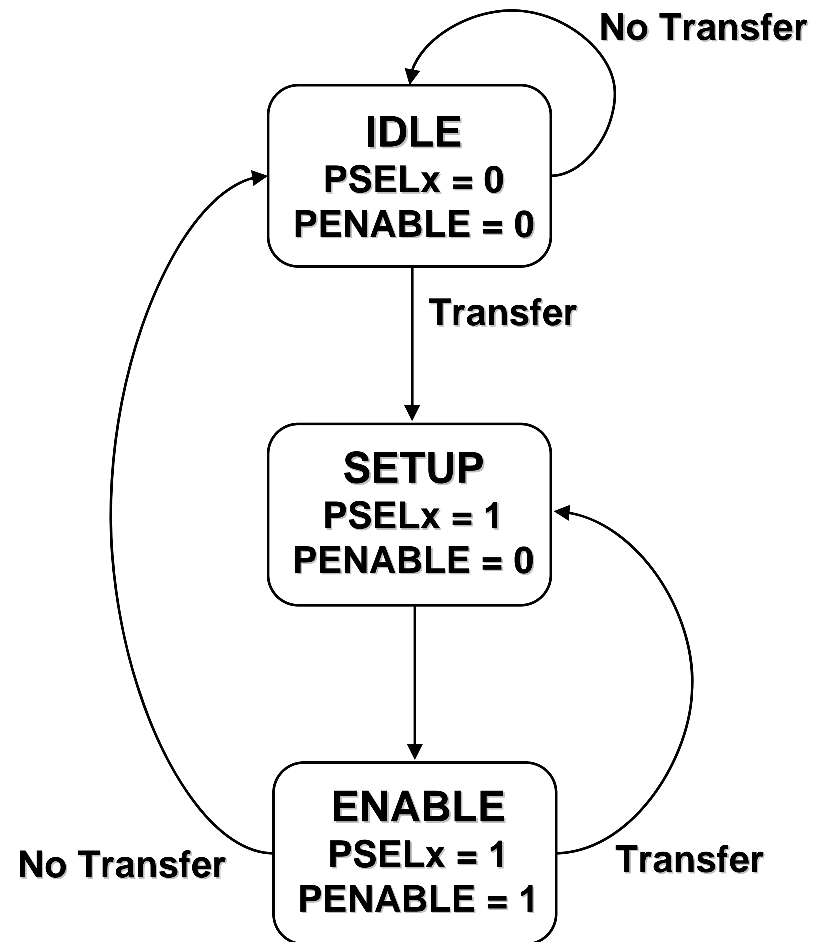
- Defines an on-chip communications standard for designing high-performance embedded microcontrollers
- Three Components:
  - Advanced High-performance Bus (AHB)
  - Advanced System Bus (ASB)
  - Advanced Peripheral Bus (APB)



APB & Its State Diagram

- APB is used to interface to any peripherals which are of low bandwidth and do not require high performance of pipelined bus interface

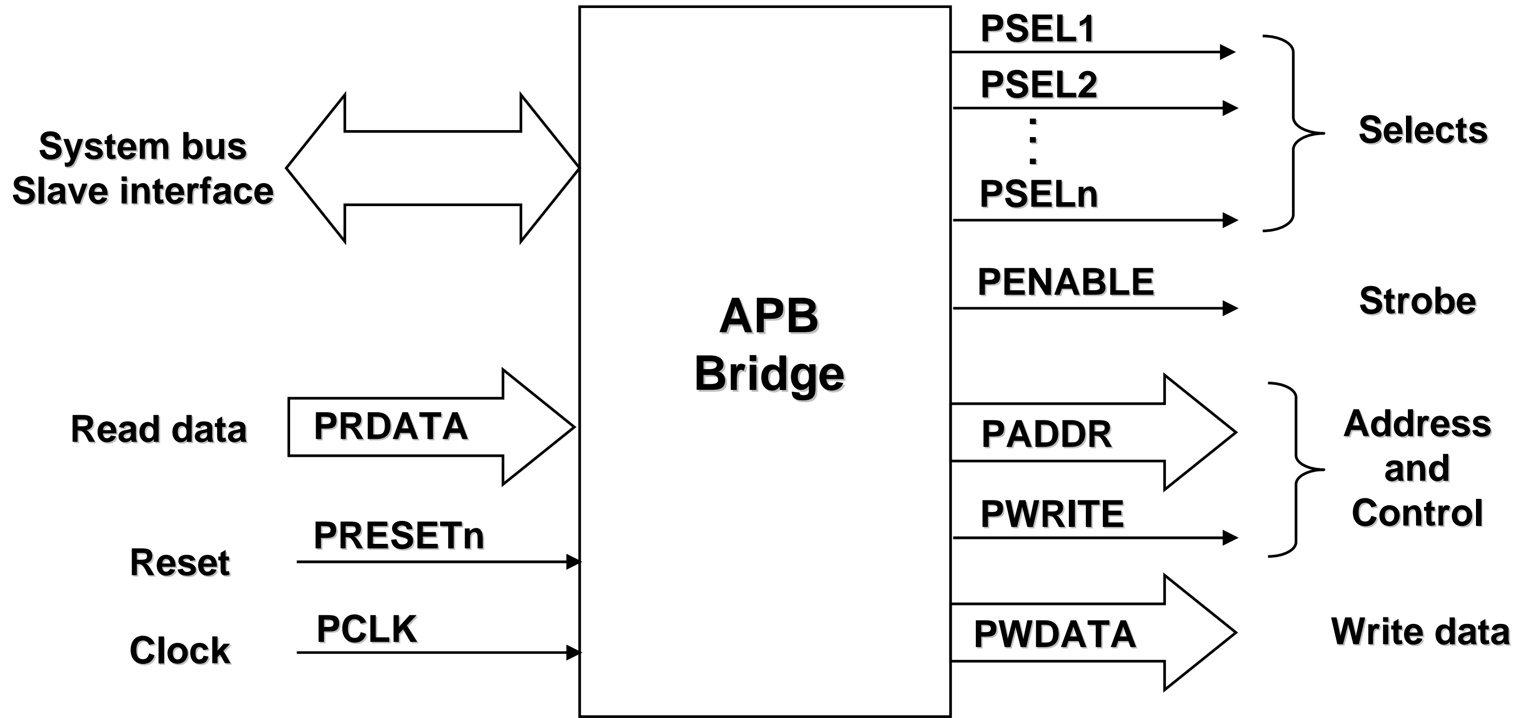
- Salient Features:
  - Low power consumption
  - Reduced interface complexity
  - Latched address & control
  - Suitable for many peripherals



APB State Diagram

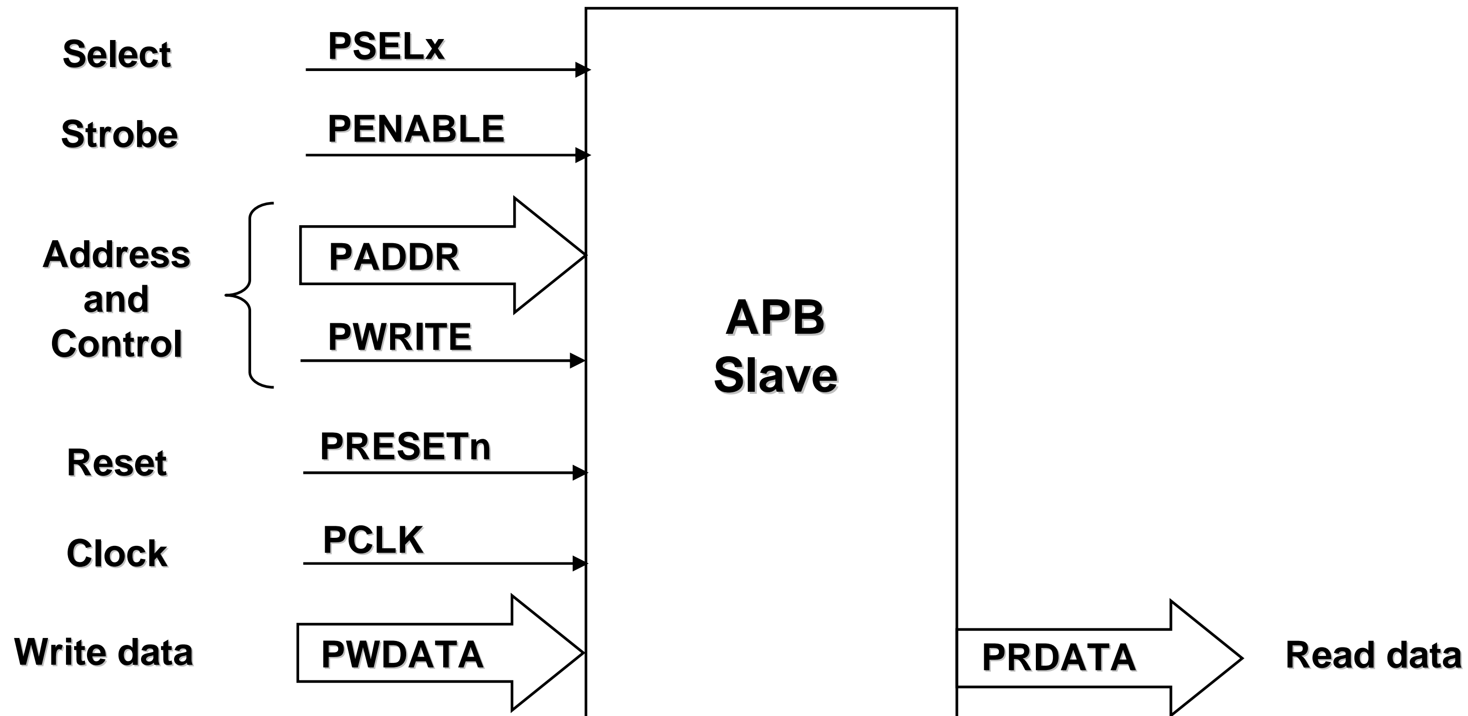
APB Bridge Interface

- APB bridge is the only bus master on the AMBA APB.
- APB bridge is also a slave on the higher-level system bus.

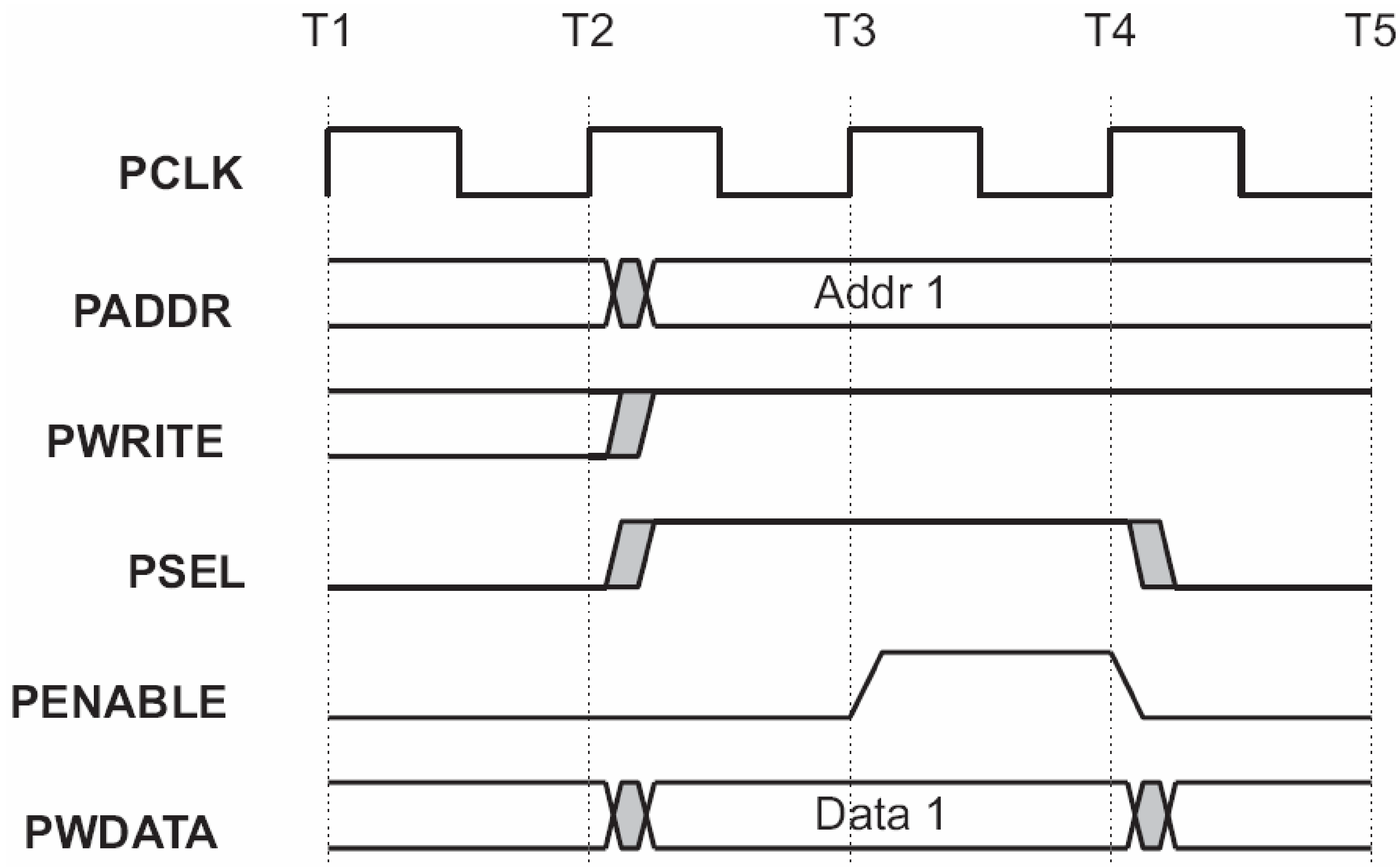


APB Slave Interface

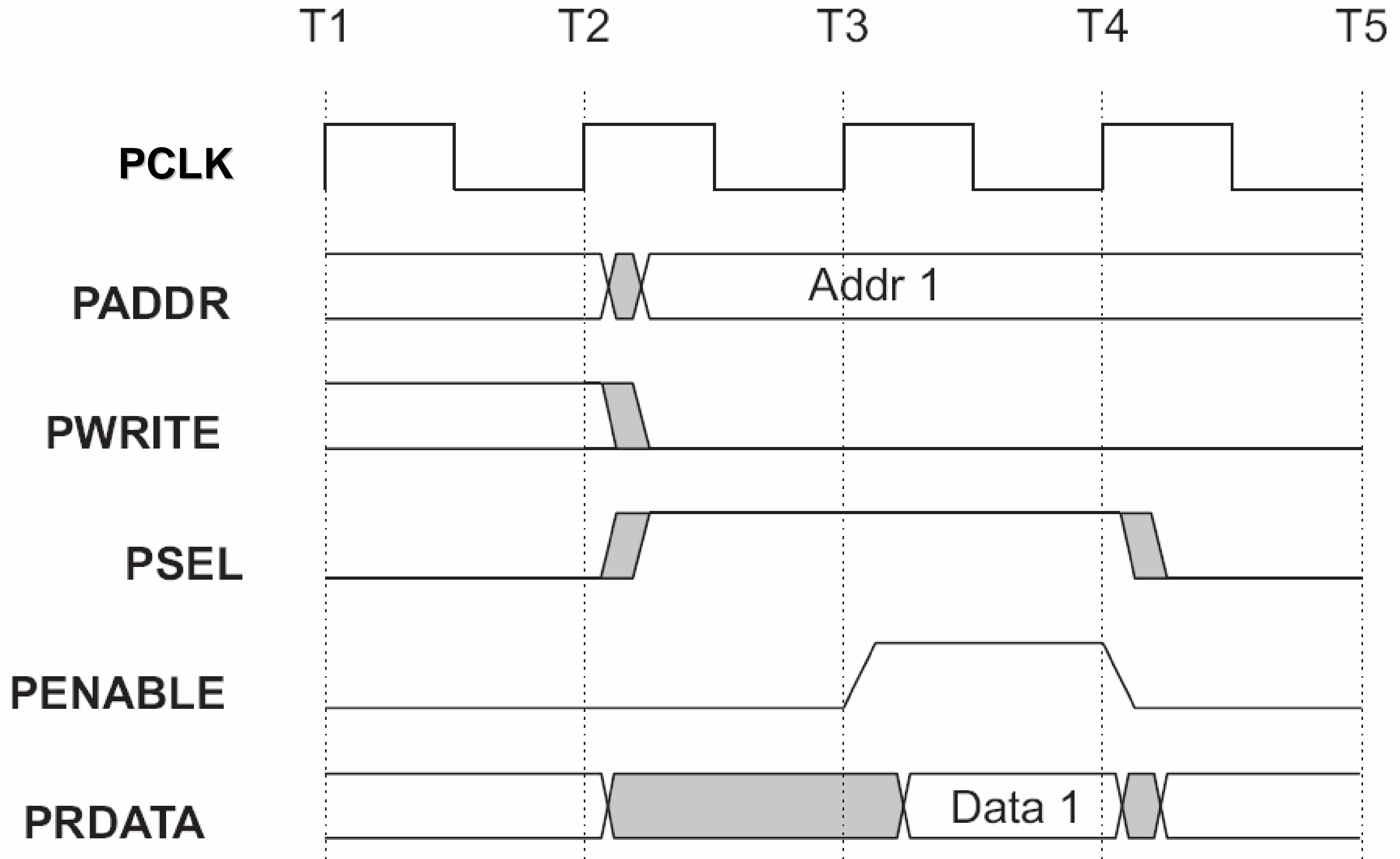
- APB slaves have a simple, yet flexible, interface.
- Exact implementation of the interface will be dependent on the design style employed and many different options are possible.



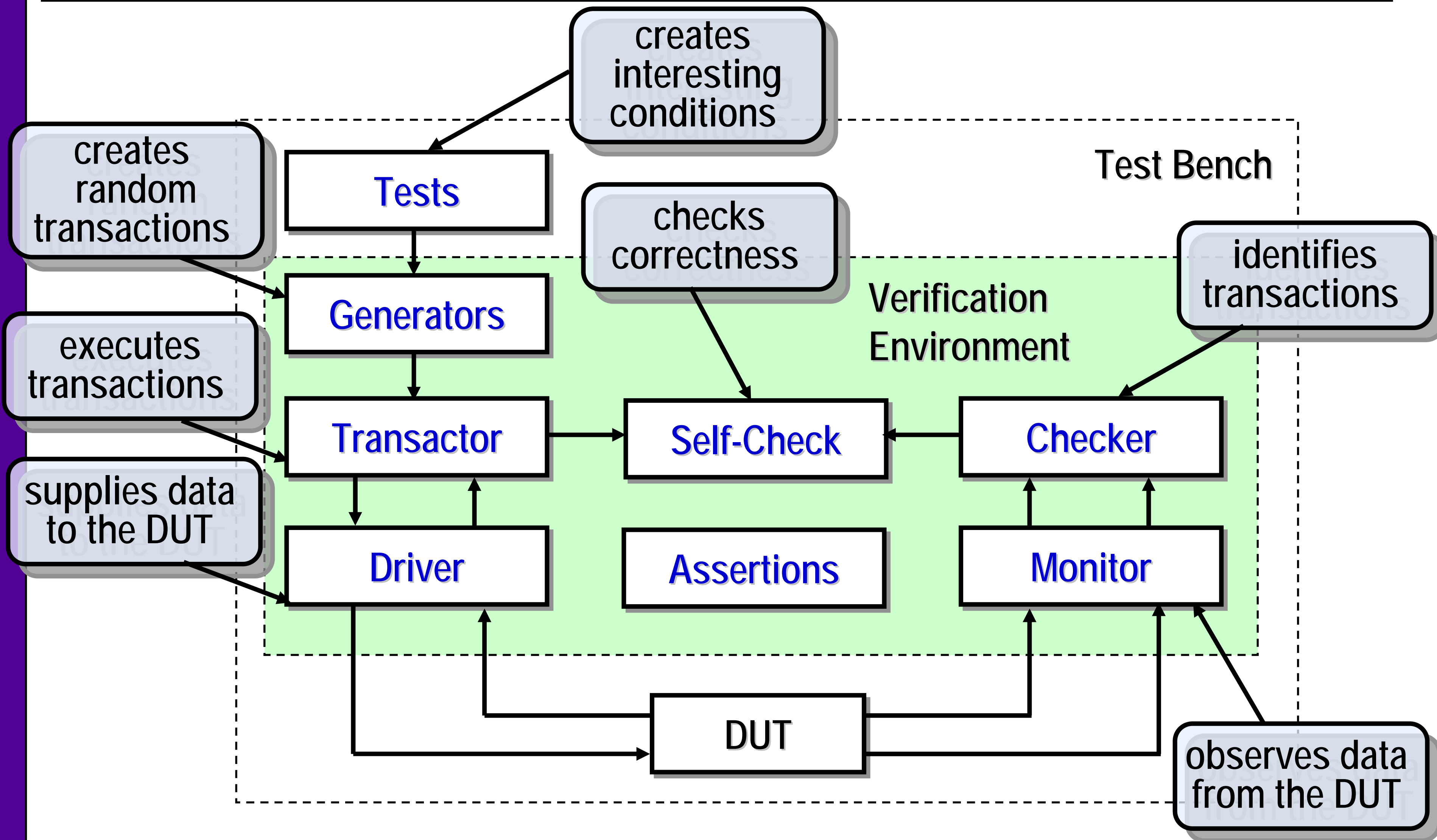
APB Write Transfer



APB Read Transfer



Layered Random Test Architecture



Environment Setup & Execute

- Top-level test bench instantiates the DUT environment, builds it and runs all steps in layered architecture by executing this environment
- The test environment structure is as follows:

```
dut_env env;                              // DUT Environment
initial begin
  env = new(mst_if, mon_if);
          // Creating environment: Master Interface and Monitor Interface
  env.build();                            // Building environment
  env.run();                              // Run all steps
end
```

Components of Environment

- **Test bench Top-level components include:**
  - APB Atomic Generator
  - APB Master
  - APB Monitor
  - Scoreboard

```
apb_cfg cfg;                     // specifies transaction
                                 // configuration
apb_trans_channel gen2mas;       // channel between
                                 // generator & master
apb_trans_channel mon2scb;       // channel between
                                 // monitor & score-board
apb_trans_atomic_gen gen;        // APB transaction generator
apb_master mst;                  // APB master
apb_monitor mon;                 // APB monitor
dut_sb scb;                      // scoreboard
```

Overall Execution Flow

Generate Configuration for Tests

```
class apb_cfg;
rand int trans_cnt;
```

Generate Random Test Scenarios

```
this.randomized_obj.randomize();
$cast(tr,this.randomized_obj.copy());
this.out_chan.put(tr);
```

Execute Individual Test Scenarios

```
in_chan.get(tr);
if ((tr.dir) == apb_trans::WRITE)
        do_write(tr);
```

Execute Commands Corresponding to Test

```
task apb_master::do_write(apb_trans tr);

…

endtask: do_write
```

Signal Layer

- Specifies DUT interface signals

```
interface apb_if(input PClk);
  logic [`APB_ADDR_WIDTH-1:0]  PAddr;
  logic PSel;
  logic [`APB_DATA_WIDTH-1:0]  PWData;
  logic [`APB_DATA_WIDTH-1:0]  PRData;
  logic PEnable;
  logic PWrite;
  logic Rst;

  /* master & monitor clocking blocks */

  modport Master(clocking master_cb);
  modport Monitor(clocking monitor_cb);
  modport Slave(input PAddr, PClk, PSel, PWData,
         PEnable, PWrite, Rst, output PRData);
endinterface
```

Command Layer

- APB master implements driver routines named `do_read()`, `do_write()` and `do_idle()`

```
task  apb_master::do_write(apb_trans tr);
  // Drive Control bus
  `APB_MASTER_IF.PAddr  <= tr.addr;
  `APB_MASTER_IF.PWData <= tr.data;
  `APB_MASTER_IF.PWrite <= 1'b1;
  `APB_MASTER_IF.PSel   <= 1'b1;

  // Assert Penable
  ##1 `APB_MASTER_IF.PEnable <= 1'b1;

  // Deassert it
  ##1 `APB_MASTER_IF.PEnable <= 1'b0;
endtask: do_write
```

Functional Layer

- Functional layer (APB master) receives the transaction generated by the scenario layer from the channel.

```
apb_trans tr;
in_chan.get(tr);
case (tr.dir)
  apb_trans::READ:   do_read(tr);
  apb_trans::WRITE:  do_write(tr);
  default:           do_idle();
endcase
```

Scenario Layer

- Scenario layer (APB atomic generator) creates individual transaction object & sends to the functional layer through channel

```
apb_trans tr;
apb_trans_atomic_gen::new(…);
        this.addr = 0; this.data = 0; this.dir=IDLE;
endfunction:new


apb_trans_atomic_gen::main(…);
                this.randomized_obj.randomize();
                $cast(tr, this.randomized_obj.copy());
                          this.out_chan.put(tr);
endfunction:main
```

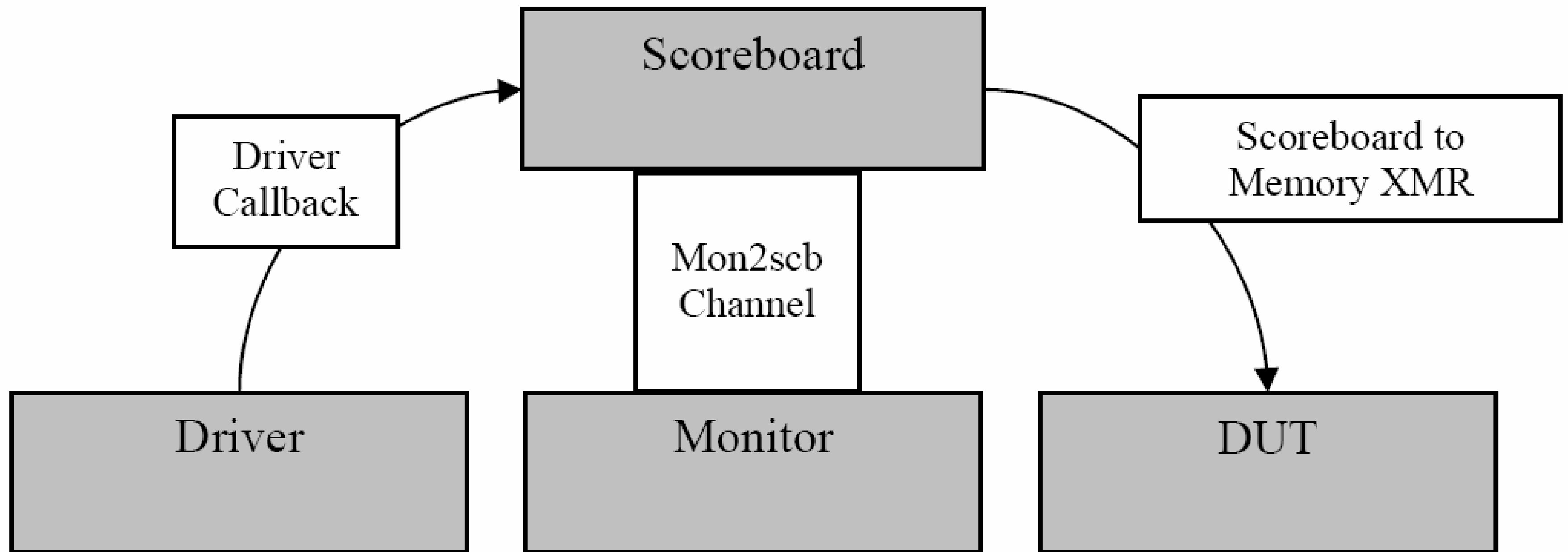
Test Layer

- env creates an object of `apb_cfg` which contains the configuration for the tests for APB VIP

```
class apb_cfg;
   rand int trans_cnt;
   constraint basic {
     trans_cnt > 5;
     trans_cnt < 10;
   }
endclass: apb_cfg

 apb_cfg cfg;
 cfg.randomize();
```

Monitor Execution Flow



Functional Layer (Monitor)

- dut_env creates an object of dut_sb which implements the checker component
- Implements check_read, check_write etc.
- scoreboard waits for a transaction to be generated then waits for the monitor to notify that this transaction occurred.
- determines the transaction correctness by applying the following:
  - Each generated WRITE transactions are stored to a register file (which acts as a reference model in this case).
  - Each generated READ transactions get their data field filled from the register file (so to provide an expected result).
  - each transactions is then compared on a first-come first-serve basis.

Functional Layer (code snippet for checker)

- Executes the following code in infinite loop:

```
mon2scb.get(mon_tr);
mas_tr = from_master_q.pop_front();
exp_data = top.m1.memory_read(mas_tr.addr);
case(mas_tr.dir)
   apb_trans::WRITE: check_write(mas_tr, mon_tr, exp_data);
   apb_trans::READ:  check_read (mas_tr, mon_tr, exp_data);
   default: `vmm_fatal(log, "Fatal error: Scoreboard
                  received illegal master transaction");
endcase

if(match >= max_trans_cnt) begin
   `vmm_verbose(this.log, $psprintf("Done scorboarding found
                                    %d matches", match));
   this.notify.indicate(this.DONE);
end
```

Command Layer (Checker)

- APB-Monitor uses callbacks to monitor the bus before and after the transaction

```
while(1) begin
  $cast(tr, randomized_obj.copy());
  // Pre-Rx Callback
  `vmm_callback(apb_monitor_callbacks ,monitor_pre_rx(this, tr));
  // Sample the bus using the apb_sample() task
  sample_apb(tr);
  // Put the trans into the output channel
  out_chan.put(tr);
  // Add a Post-Rx Callback. Typically for Coverage or Scoreboard
  `vmm_callback(apb_monitor_callbacks ,monitor_post_rx(this, tr));
  `vmm_debug(log, tr.psdisplay("Monitor ==>"));
end
```

AMBA APB Property Set

- **PSEL:**
  - If PSELx is LOW for some slave x in the present cycle (1ST) and in the next (2nd) cycle it goes HIGH, it must be also HIGH in the next (3rd) cycle.
  - At a time only one PSEL can be high i.e. only 1 slave can be selected at a time.

- **PENABLE:**
  - If PENABLE is HIGH in the present cycle, it must go LOW in the next cycle.

AMBA APB Property Set (contd..)

- **PSEL & PENABLE:**

  - If PSELx is LOW for some slave x in the present cycle (1st) and in the next cycle (2nd) it becomes HIGH then one more cycle later (3rd) PENABLE must also be HIGH.

  - If all of the PSEL is LOW in the present cycle (1st) then in the same cycle (1st) & also in the next cycle (2nd) PENABLE must also be LOW.

  - If PENABLE is HIGH (1st) and in the next cycle (2nd) PSEL is HIGH, then one more cycle (3rd) later PSEL & PENABLE are both HIGH.

AMBA APB Property Set (contd..)

- **PENABLE & PWDATA:**
  - If PENABLE is HIGH in the present cycle then PWDATA will hold the same value as the previous cycle.
- **PSEL & PWRITE:**
  - If PWRITE changes one of the PSEL must be HIGH.
- **PENABLE & PWRITE:**
  - If PENABLE is HIGH in the present cycle then PWRITE will hold the same value as the previous cycle.
- **PENABLE & PADDR:**
  - If PENABLE is HIGH in the present cycle then PADDR will hold the same value as the previous cycle.