SPLATTERS COM6502

SYNTHESIZABLE PLATFORM FOR TERRIFIC SIMULATION

Virtual Platform (VP)

- Simulatable abstraction of a complex design
  - Composed of multiple **IPs**
    - **Design reuse!**
  - High performances
  - Adequate accuracy
- Enable **HW-SW Codesign**
  - Reducing time-to-market
  - Powerful debug and verification features
- Choice of **abstraction level** is fundamental
  - **Compromise** between accuracy and complexity

Virtual Platform (VP)

- Typical VP design components:
  - System IP models
  - Memory models → how memory works
  - Processor models → operations
    - A compiler is necessary to write software for the VP!
    - **Cross-compilers enable the generation of platform-native executables on an external machine**
  - Communication models
  - Accessing host interfaces in the Virtual Platform
    - Interfaces for communicating with external devices
  - Debugger interface support

COM6502-Splatters

- The platform includes:
  - CPU MOS 6502 (1975)
    - Vic 20, Commodore 64, Nintendo Entertainment System (NES), Apple II, Atari 800, Terminator
    - 16 bit addressing (16KB ROM, 16KB RAM),
    - 8 bit data width
  - Memory
    - ROM in one single bank
    - RAM splitted in 8 different blocks to enable multi read/write operations
    - Clock divider
      - manage MMIO operation between Peripherals, Memory
      - Manage multiple write on same cell between CPU and MMIO Interface
  - BUS ARM APB (Advanced Peripheral Bus)
    - Supports up to 8 peripherals
  - IO Module
    - Used to request or send data out from the platform
  - Simple Integer Multiplier

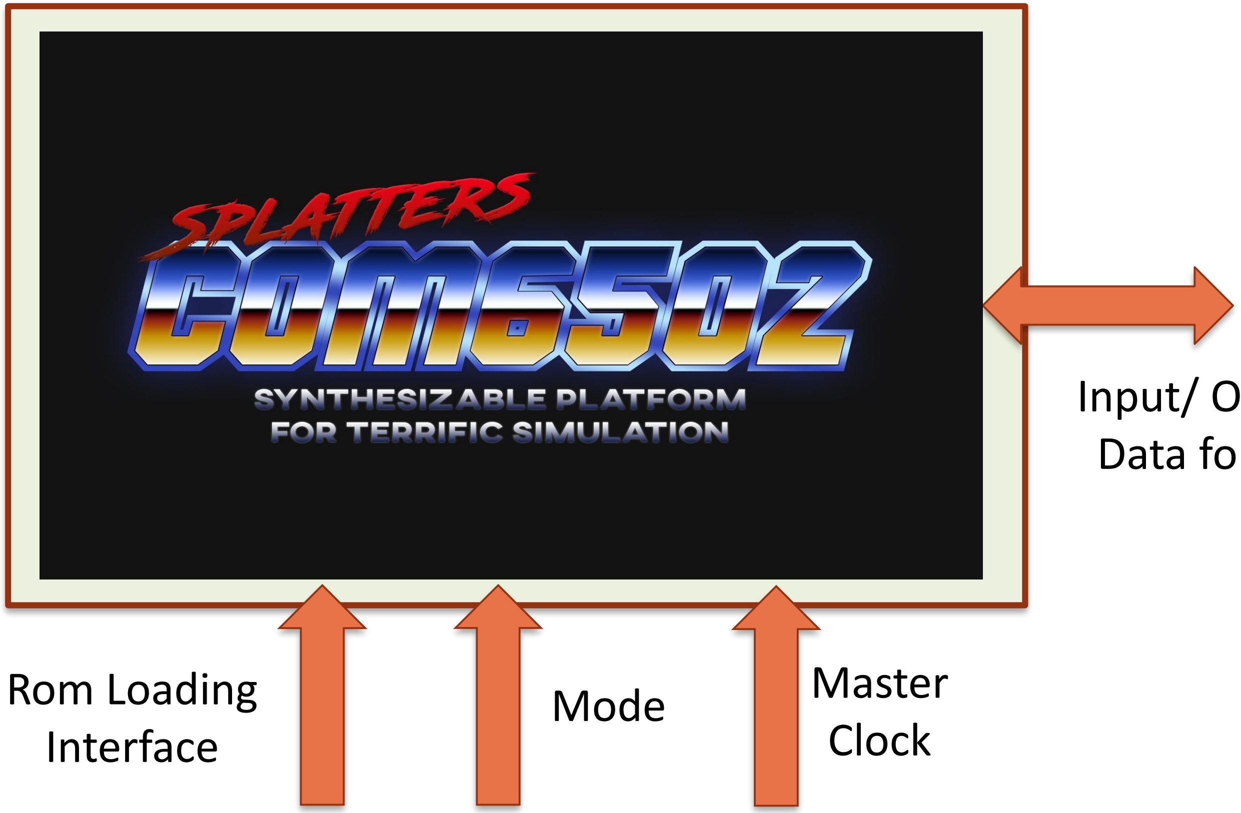
COM6502-Splatters

- Unzip com6502-splatters.tar.gz
  - Three directories
    - platform/
      - Contains the description of all Com6502-Splatters components including a tb
    - cc65/
      - Cross-compiler binaries (UNIX)
    - application/
      - Main example
      - MMIO functions → implementation of Memory Mapped I/O → data read and stored is contained in the main memory in a specific range of addresses
      - Drivers for Multiplier and IO Module

Splatters - Top Level

top_level.v

- Two different mode available
  - Load cross-compiled SW
  - Run SW with I/O interactions



Input/ Output
Data for SW

Rom Loading
Interface

Mode

Master
Clock

COM6502-Splatters

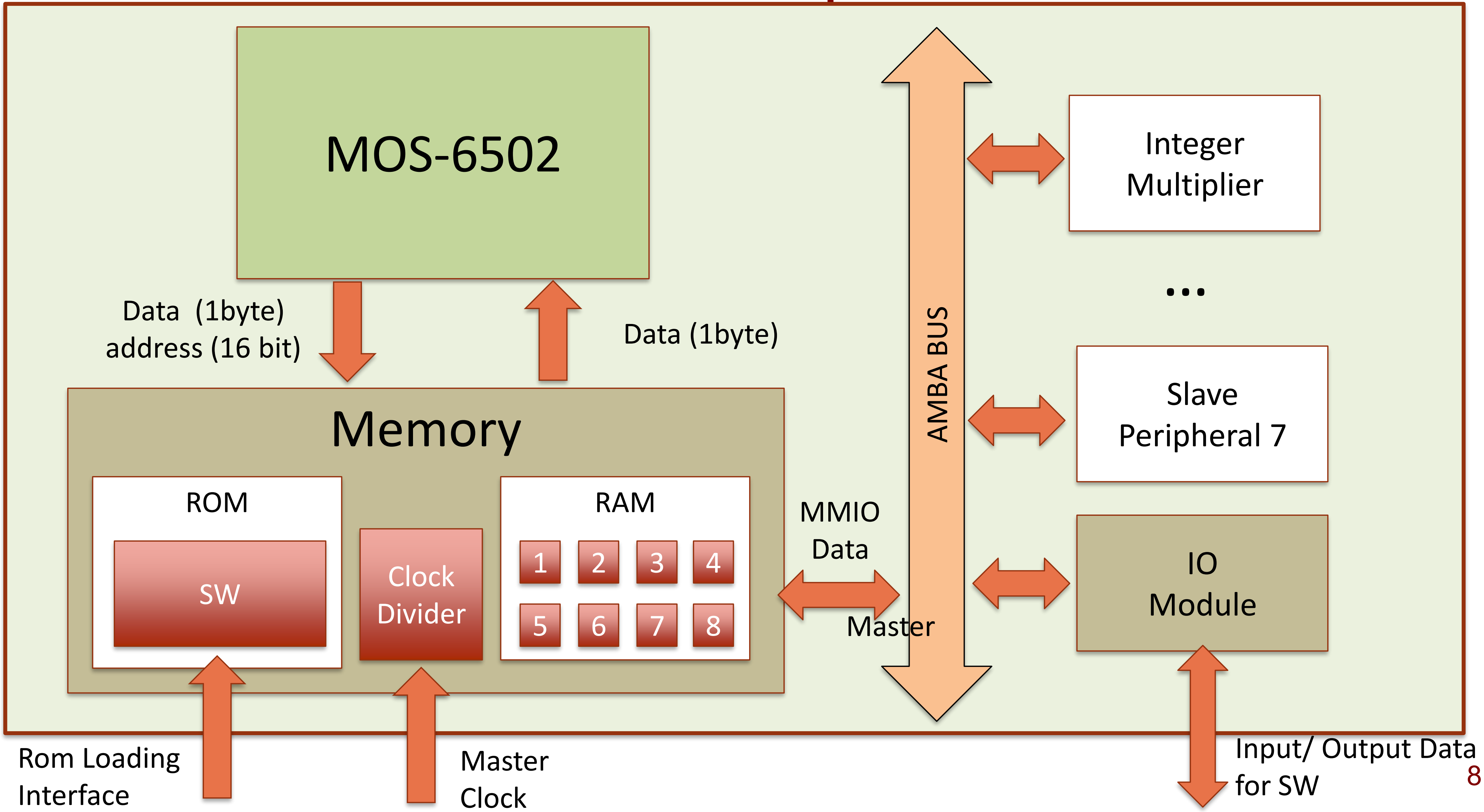
Top Level

	Port	Description	Causality	Data width
	clk	Platform clk	Input	1
	rst	Platform Reset	Input	1

	Mode	Mode = 0 Load Rom
		Mode = 1 Run Rom SW
			Input	1

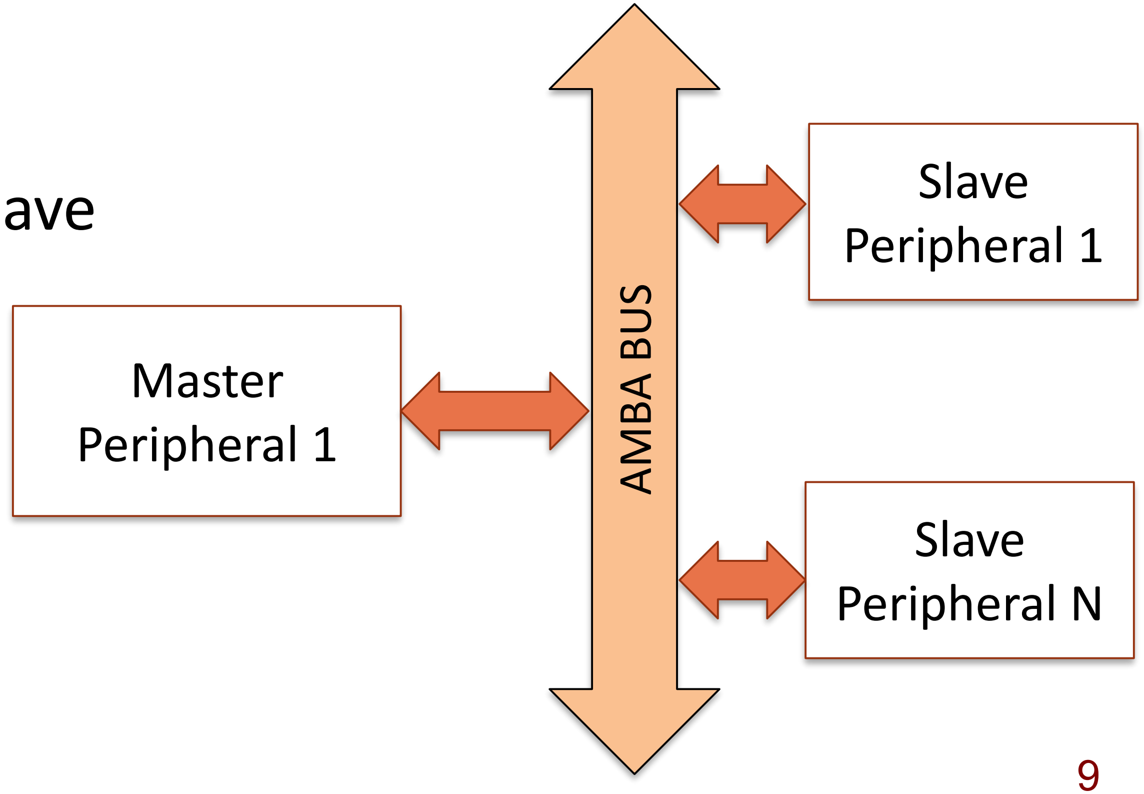
Rom Loading Interface	we_n_in	SW Write enable on negative edge	Input	1
	sw_din	SW data input	Input	8
	sw_addr	SW address	Input	14
Input/ Output Data for SW	dout	Output Data from Platform	Output	32
	dout_rdy	Output Data Ready from Platform	Output	1
	din_req	Data Request from Platform	Output	1
	din	Input Data for Platform	Input	32
	din_rdy	Input Data Ready for Platform	Input	1

COM6502- Splatters



AMBA Advanced Peripheral Bus (APB) Protocol

- The ARM Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs.
- Specifies two actors
  - Master : control other peripherals
  - Slave: peripheral controlled by Master
- Defines specific interfaces for Master/Slave
- Defines how Master slave interacts



AMBA Advanced Peripheral Bus (APB) Protocol

Master Interface

(Bus Perspective)

Port	Description	Causality	Data Width
apb_master_presetn	Reset the peripheral	Input	1
apb_master_paddr	Address for peripheral	Input	32
apb_master_pselX	Peripheral Selector (X = 1..8)	Input	1
apb_master_penable	Enable Peripheral	Input	1
apb_master_pwrite	Read/Write Operation	Input	1
apb_master_pwdata	Data for Peripheral	Input	32
apb_master_pready	Data Ready from Peripheral	Output	1
apb_master_prdata	Data From Peripheral	Output	32

AMBA Advanced Peripheral Bus (APB) Protocol

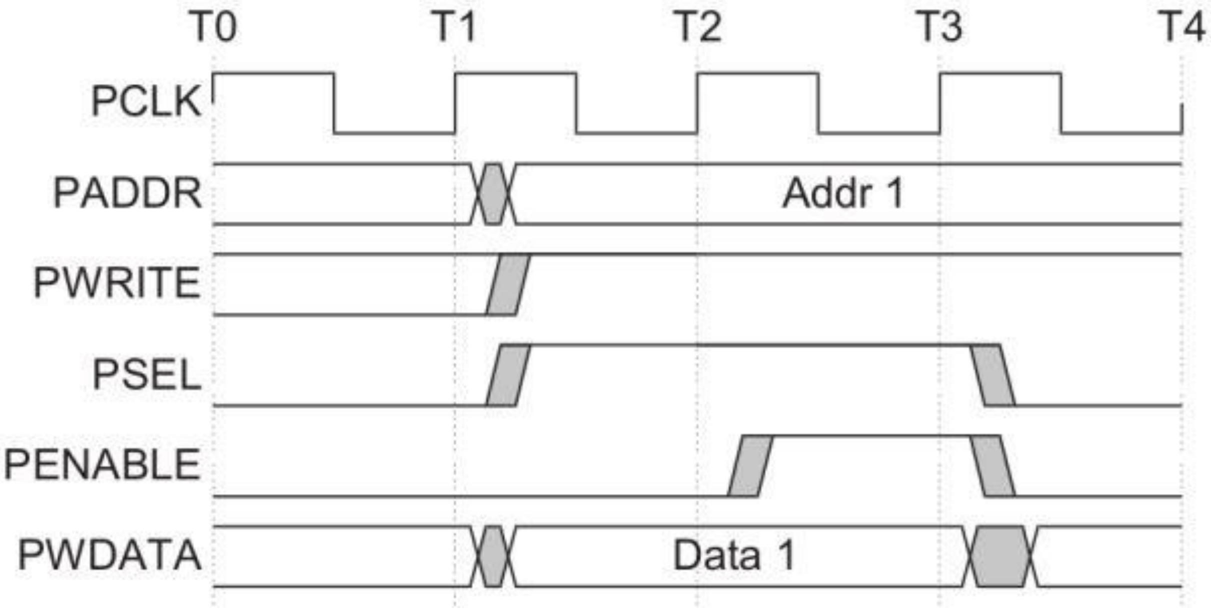
Slave Interface

(Bus Perspective)

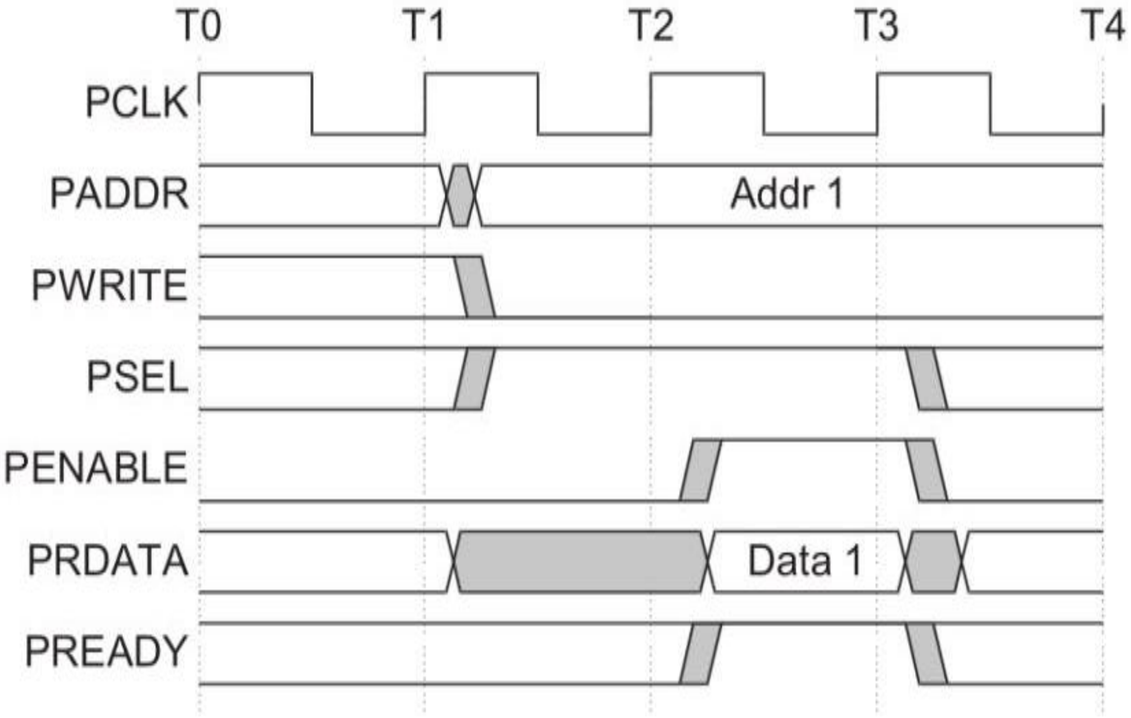
Port	Description	Causality	Data Width
apb_X_pclk	Peripheral Clock	Output	1
apb_X_presetn	Reset	Output	1
apb_X_paddr	Data Address	Output	32
apb_X_psel	Peripheral Selected	Output	1
apb_X_penable	Enable Peripheral	Output	1
apb_X_pwrite	Read/Write Operation	Output	1
apb_X_pwdata	Data from Master	Output	32
apb_X_pready	Data Ready for Master	Input	1
apb_X_prdata	Data For Master	Input	32

AMBA Advanced Peripheral Bus (APB) Protocol

Write Transfer

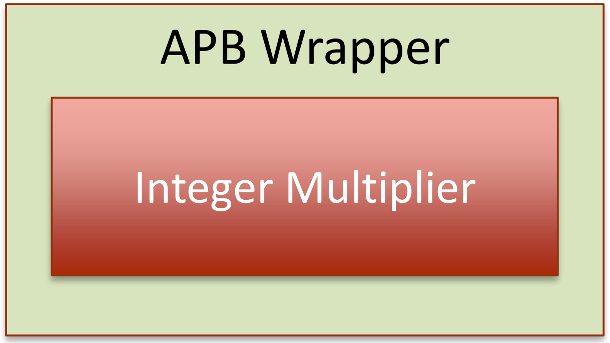


Read Transfer



Integer Multiplier

- Simple Integer Multiplier



```
ENTITY multiplier IS
 PORT (
  clk    : in std_logic;
  din_rdy: in std_logic;
  op1    : in std_logic_vector(15 downto 0);
  op2    : in std_logic_vector(15 downto 0);
  result : out std_logic_vector(31 downto 0);
  dout_rdy  : out std_logic);
END multiplier;
```

```
ARCHITECTURE multiplier OF multiplier IS
BEGIN
    PROCESS (clk)
    BEGIN
        IF ( clk'event and clk = '1' ) THEN
            IF ( din_rdy = '1' ) THEN
                result   <= (op1 * op2);
                dout_rdy <= '1';
            ELSE
                result   <= ( others => '0' );
                dout_rdy <= '0';
            END IF;
        END IF;
    END PROCESS;
END multiplier;
```

Integer Multiplier

- APB Wrapper

```
ENTITY multiplier_apb_wrapper IS
PORT (
  pclk    : in std_logic;
  presetn : in std_logic;
  paddr   : in std_logic_vector(31 downto 0);
  psel    : in std_logic;
  penable : in std_logic;
  pwrite  : in std_logic;
  pwdata  : in std_logic_vector(31 downto 0);
  pready  : out std_logic;
  prdata  : out std_logic_vector(31 downto 0));
END multiplier_apb_wrapper;
…
```

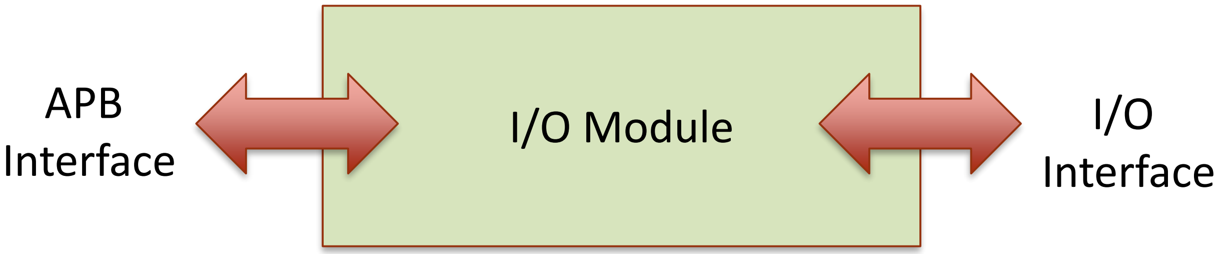
```
ARCHITECTURE multiplier_apb_wrapper OF
multiplier_apb_wrapper IS
…
BEGIN
clk     <= pclk;
op1     <= pwdata(15 downto 0);
op2     <= pwdata(31 downto 16);
din_rdy <= penable;

pready <= dout_rdy;
prdata <= result;

multiplier_0 : multiplier
PORT MAP(
        clk     => clk,
        din_rdy => din_rdy,
        op1     => op1,
        op2     => op2,
        dout_rdy=> dout_rdy,
        result  => result
        );
END multiplier_apb_wrapper;
```

I/O Module

- Allows external interaction with the platform
  - Expose output data via SW
  - Request external data for the SW



I/O Interface

Port	Description	Causality	Data
dout	Output Data from Platform	Output	32
dout_rdy	Output Data Ready from Platform	Output	1
din_req	Data Request from Platform	Output	1
din	Input Data for Platform	Input	32
din_rdy	Input Data Ready for Platform	Input	1

+

APB Slave Interface

COM6502 – Splatters

Software

- SW can be written in C Language
  - CC65 Cross Compiler for MOS 6502 CPU
    - https://www.cc65.org/
    - No support for float/double types
      - IEEE 754 didn’t exists when MOS 6502 was developed!
    - Int datatype is only 1 byte

cc65
- the 6502 C compiler

- ESD Team Developed a MMIO library in C
  - Allows to easily write simple drivers for Custom Slave Peripherals

- cc65 precompiled in com6502-splatter.tar.gz or source code from Github
  - https://github.com/cc65/cc65
  - Checkout this commit: 582aa41f2a702ff477a00a5d69a794390a13b544

COM6502 – Splatters

Software

- application/inc/mmio.h
  - Contains functions to interact with APB Master interface
  - set_pwdata,set_psel,set_penable, get_prdata, get_pready ecc…
- application/inc/routines.h
  - Contains Multiplier and IO Module drivers

Multiplier Integer Simple Driver

```
uint32_t mul(uint16_t op1, uint16_t op2)
{
   // Prepare the variable
   uint32_t result = 0x0000;
   // Prepare the data to send.
   set_pwdata_16(op1, op2);
   // Select the peripheral.
   set_psel(PSEL1);
   // Enable the operation.
   set_penable(1);
```

```
// stay here untile prdata_rdy is ready.
while (get_pready() == 0) __asm__("nop");
// Get the result.
result = get_prdata();
// Disable the operation.
set_penable(0);
// Disable data stream from the bus.
set_psel(NO_PSEL);
// Return the result.
return result;
}
```

COM6502-Splatters

Software

- application/src/main.c

```
int main()
{
        uint16_t op1    = 5;
        uint16_t op2    = 2;
        uint32_t result = 0;

        result = mul(op1, op2);              // Call Multiplier Driver

        io_write(result);                    // Write the result on IO Module

        op2    = (uint16_t) io_read();       // Read new data from IO Module

        result = mul(op1, op2);              // Execute multiplication with new op2

        io_write(result);                    // Write result on IO Module

        return 0;
}
```

Hands on

1. Create a new Vivado Project importing all the components platform/
2. Export CC65_DIR global variable with the path to cc65/ directory
   - `> export CC65_DIR=<your_path>/cc65/`
3. Move into application/ directory
4. Launch compilation
   - `> make`
5. Import the cross compiled SW (application/bin/rom.mem) in Vivado
   - Add Sources
6. Run simulation!
   - run 1ms

Lecture Assignment

- Connect the Root module to COM6502-Splatters Platform
  - Wrap the design with APB Slave interface
  - Instantiate and bind the design to APB Bus
- Write a SW Driver for the Root
  - Add new function to routines.c file
  - Request at least one of four operators via I/O Module

- **Detail the choices you made in the Report!** → Both for wrapping and writing the driver → 1-2 pages