Embedded Programming: Memory-Mapped I/O

Interacting With Hardware Devices From Software

By Jason Agron

Memory-Mapped?

- Bus-based systems have an address space.
  - Often times called “memory space”.
- Specific devices are associated with specific address ranges.
- Addresses mean different things for different types of devices.
  - Memory - individual storage locations.
  - IP cores - register based storage/command locations.
- *How does a CPU access a bus?*
  - *HINT - instruction type?*

What is Memory-Mapped I/O?

- Memory-mapped I/O is the process of either…
  - Sending output to a memory-mapped location.
  - Getting input from a memory-mapped location.
- The CPU is able to do this using…
  - LOAD and STORE instructions.
  - Essentially, performing writes and reads on the bus.
  - This is an easy and universal way for a CPU to communicate with other devices.
- Pseudo code:
  - inData = LOAD(address,offset);
  - STORE(address,offset,outData);

What Can I Use M.M. I/O For?

- Memory devices:
  - Storing/retrieving data.
- Peripheral devices:
  - Sending/receiving data.
  - Controlling device (modes, setup, etc.).
- HW Accelerators:
  - Setting up and controlling the execution of specialized HW circuits.
- External devices:
  - Reading switch and button settings.
  - Controlling LEDs.
  - The list goes on and on…

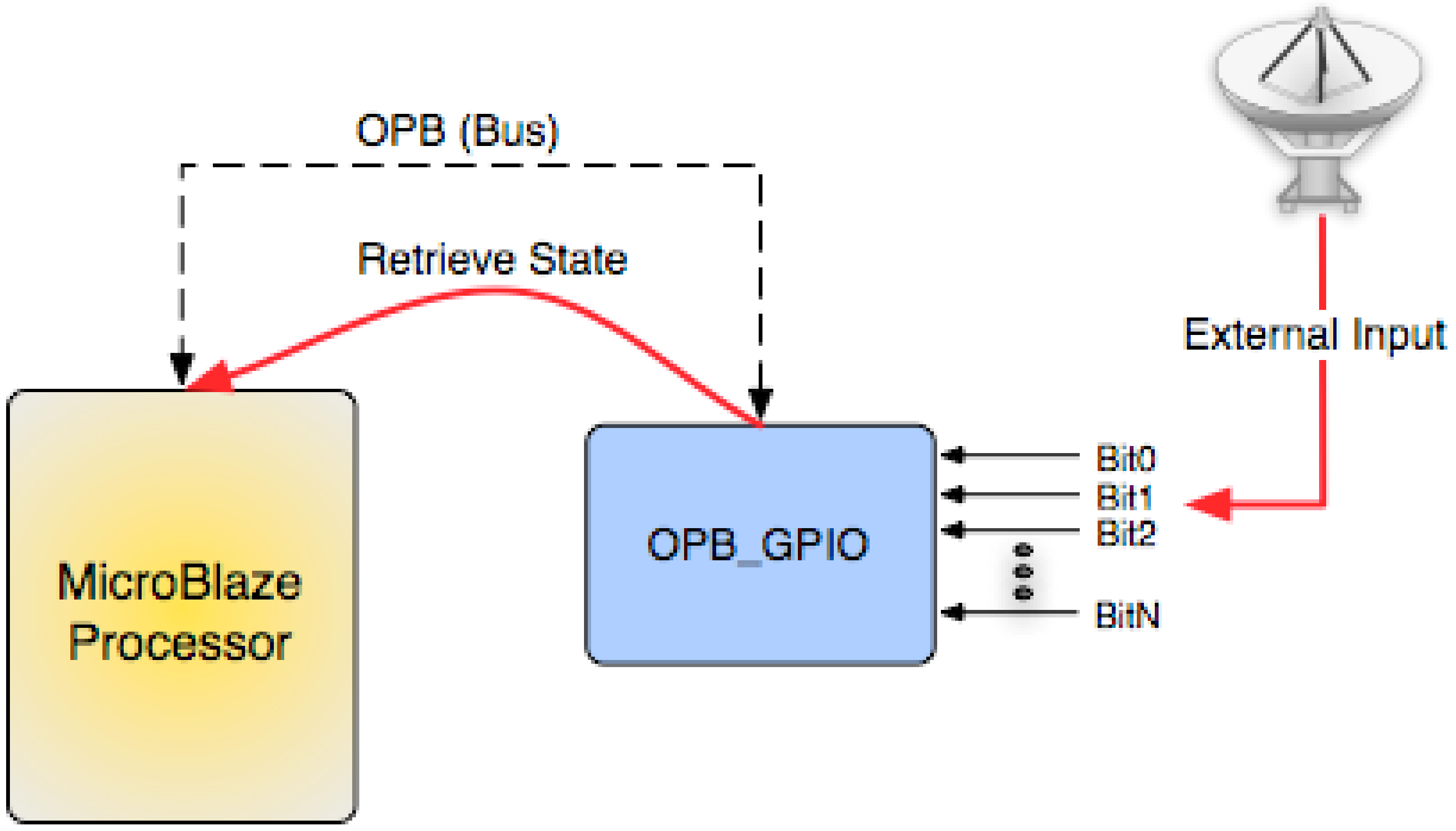
Goal Of This Project

- To create a system capable of interacting with the switches, buttons, and LEDs on the XUP board.
- Create a SW application that controls the LEDs based on the “state” of the on-board buttons and switches.
- This can all be done via the OPB_GPIO devices for each peripheral.

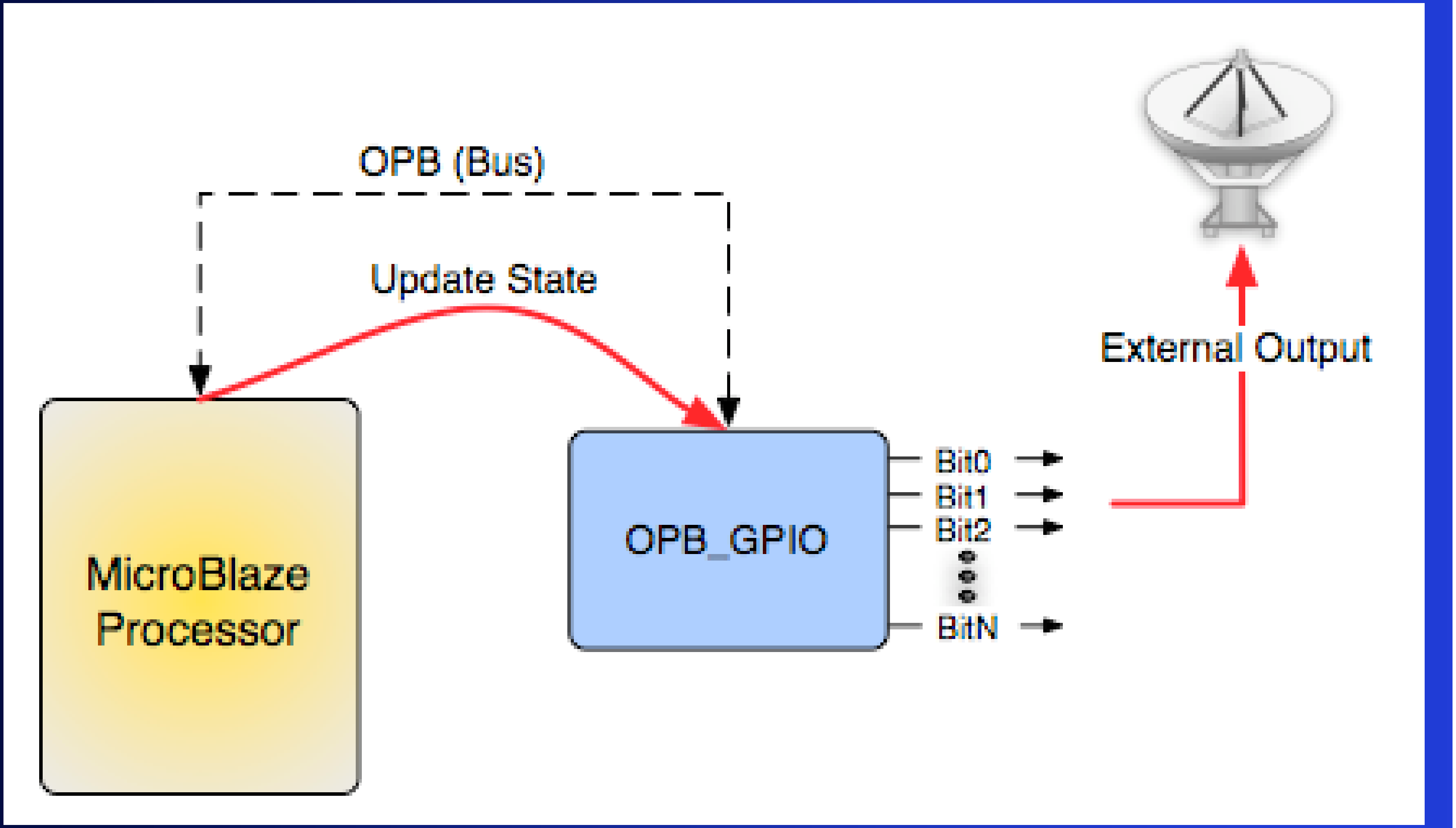
Controlling Peripherals: OPB_GPIO

- An OPB-based IP Core.
- GPIO = General Purpose I/O.
- A simple register interface that can function as either an input or output register.
    - Input Mode:
        - Register value is read by software.
        - State is inputted on HW signals.
    - Output Mode:
        - Register value is written by software.
        - State is outputted on HW signals.

GPIO: Input Mode



GPIO: Output Mode



OPB_GPIO Register Set

- Each GPIO can be configured to have 1 to 2 ports.
- GPIOx_DATA:
  - Data register for GPIO port x.
  - Used to read input ports and write to output ports.
- GPIOx_TRI:
  - 3-State register for GPIO port x.
  - Used to dynamically configure the direction of a port.

OPB_GPIO Settings

- GPIOx_DATA Settings:
  - Input Mode -
    - READ - Reads value on input pins.
    - WRITE - No effect.
  - Output Mode -
    - READ - Reads value in data register.
    - WRITE - Writes value to data register and output pins.

- GPIOx_TRI Settings:
  - Each bit can be individually programmed as input or output.
  - 0 = Output Mode.
  - 1 = Input Mode.

OPB_GPIO Memory-Map

- GPIO Channel (Port) 1:
  - GPIO1_DATA -
    - BASEADDRESS + 0x00
  - GPIO1_TRI -
    - BASEADDRESS + 0x04
- GPIO Channel (Port) 2:
  - GPIO2_DATA -
    - BASEADDRESS + 0x08
  - GPIO2_TRI -
    - BASEADDRESS + 0x0C

- *Why is each register separated by an offset of 0x04?*
  - *HINT - What is the bit-width of each register?*

How Do I Access GPIO Registers in SW?

- Programming constructs must be used that allow one to read/write specific addresses.
- In assembly:
  - LOADs and STOREs.
- In C:
  - Done with pointers.

XGPIO Libraries

- Xilinx has C libraries that provide functions to control OPB_GPIO devices.
- Eliminates the need to explicitly use pointers.
  - Pointer/memory operations are now done within the Xilinx-provided functions.
- An example of these functions will be provided.

Pointers!

- Pointers:
  - A programming construct.
  - Used to “point” to a specific location.
    - Often times memory.
  - Features:
    - A location to point to (address).
    - Something being pointed at (data).

Pointer Example

- A pointer to an integer stored at location 0x5000000.
  - *volatile int *myPtr = (int*)0x5000000;*
  - *What does the "volatile" keyword do?*
- Writing data to the location:
  - **myPtr = <newData>;*
- Reading data from the location:
  - *dataAtLocation = *myPtr;*
- Changing the location being pointed at:
  - *myPtr = <newLocation>;*
- *What is the "*" doing????*

Application Hints

- First create a new application.
- Verify system operation with a “Hello World” program.
- Add pointers for all required registers.
  - Print out pointer values to check that they “point” to the “right” addresses.
- Create a small control-loop program to repetitively read the state of the buttons.
  - Update the state of the LEDs on each iteration.
  - Use print() statements to debug and/or display values.