

**Hardware – Software  
Co-Simulation with System C  
By  
Richard Ruigrok  
Qualcomm Incorporated**

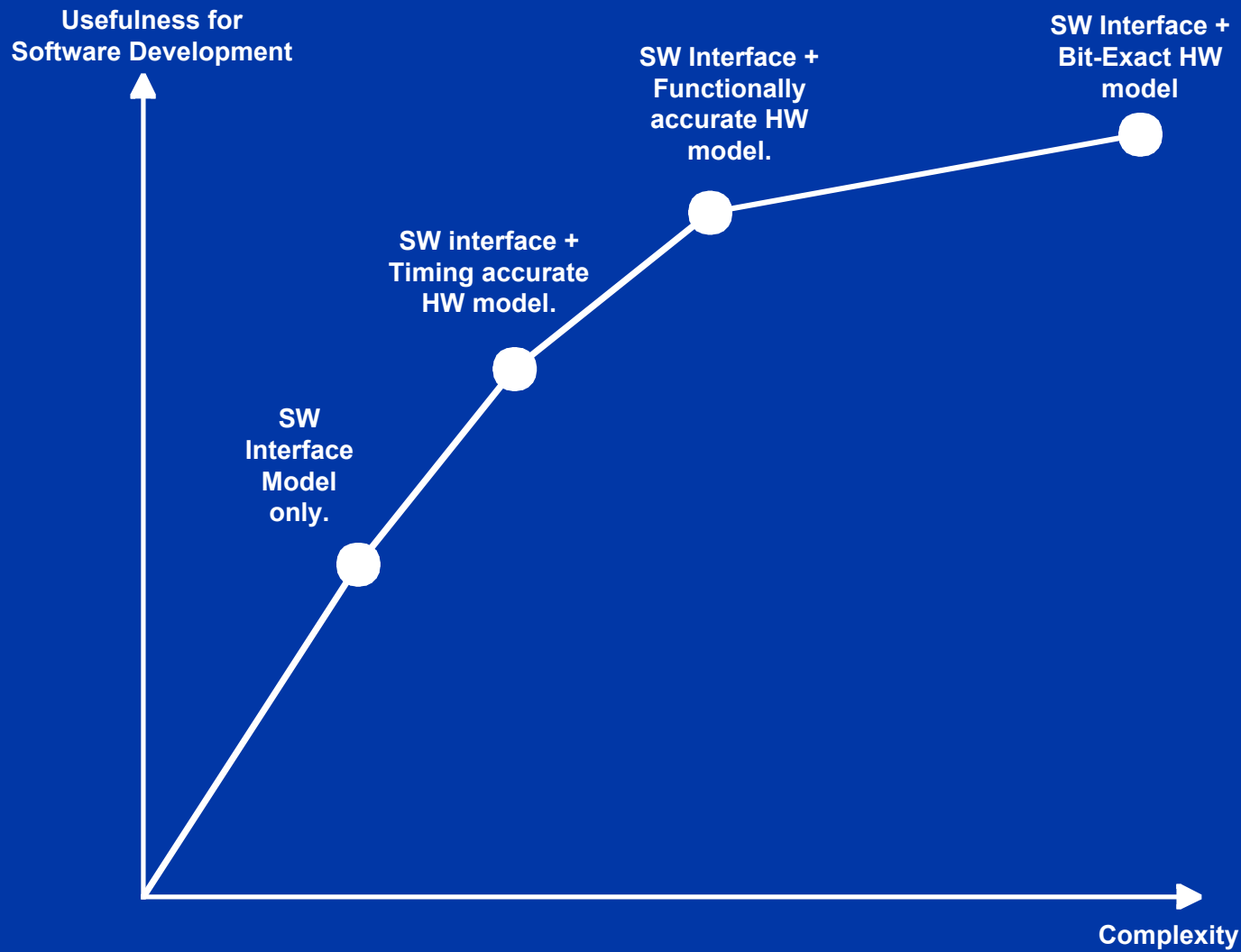
# Objectives

- Provide an overview of Co-Simulation model types, how they work and how they help with software development.
- Identify basic building blocks necessary to build a Co-Simulation model, including code examples.
- Demonstrate Co-Simulation using a coprocessor hardware model with pipelining.
- Show how to integrate coprocessor model and software into a Co-Simulation model.
- Show Co-Simulation results from coprocessor model using two different Co-Simulation models.

# Overview of Co-Simulation Models

- There are many different models and methods, two basic methods are outlined in this presentation:
  - Co-Simulation on WS (Workstation)
    - Software compiled for native WS processor.
    - Software timing is not possible.
    - Software runs very fast.
    - Attach to HW model of varying complexity.
  - Co-Simulation with TPS (Target Processor Simulator)
    - Software compiled for target processor.
    - Requires a third-party TPS model.
    - Provides software and hardware execution time, accuracy is TPS dependent.
    - Can be used to optimize both hardware and software.
    - Attach to same HW model of varying complexity.

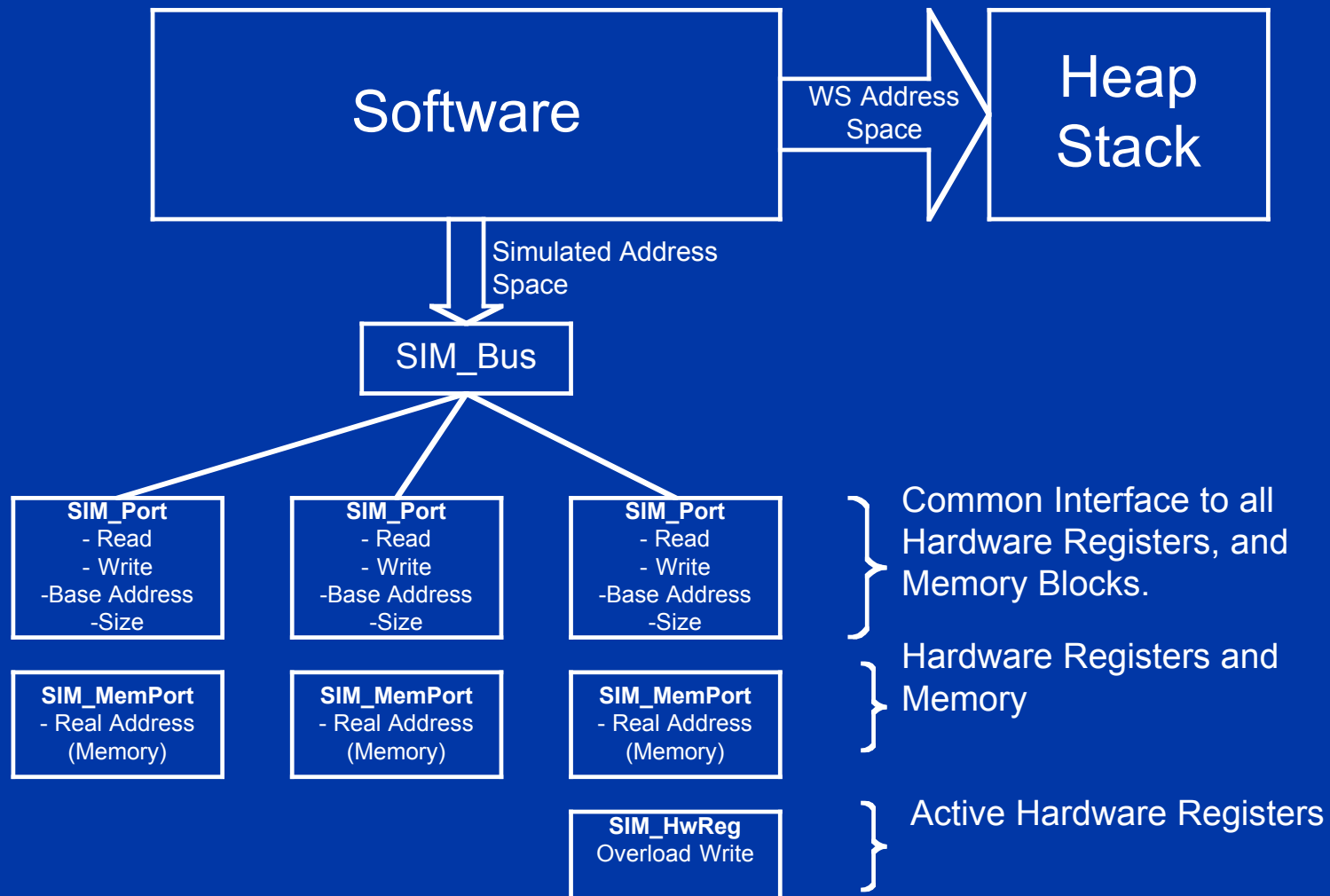
# Overview of Co-Simulation Models



# Building Blocks for Co-Simulation on WS

- Code must compile and run on WS.
- Any access to hardware must be re-directed to simulated bus (SIM\_Bus).
- Software Heap and Stack memory provided by WS.
- Hardware memory is implemented by a simulated memory on WS (SIM\_MemPort)
- Hardware active blocks (registers, etc) are also SIM\_MemPorts, with active processing components.
- Use simplest bus architecture, arbitration and wait states can be added later.
- Tip: Use Macros to easily re-direct hardware accesses.

# Baseline Software and Simulated Bus



# Software Interface Code Examples

## Macros to read, write to hardware.

```
#if defined(SIM_MODE)
extern void    SIMBUS_Write(const uint32 address, const uint32 value );
extern uint32 SIMBUS_Read(const uint32 address);

    #define MACROS_WRITE(addr, data)                \
                SIMBUS_Write( ( uint32 )addr, ( uint32 )data)

    #define MACROS_READ(addr)                       \
                SIMBUS_Read(( uint32 )addr)

#else
    #define MACROS_WRITE(addr, data)                \
                (*(volatile uint32 *) (addr)) = ((uint32) (data))

    #define MACROS_READ(addr)                       \
                (*(volatile uint32 *) (addr))
#endif
```

# Simulated Memory (SIM\_Mem)

```
class SIM_MemPort : public SIM_Port
{
public:
    virtual ~SIM_MemPort(){};
    SIM_MemPort(uint32 baseAddress = 0,
                uint32 size = 0
                )
    {
        BaseAddress=baseAddress;
        SizeWords=((size + 3) / 4);
        MemPtr = new uint32[ SizeWords ];
        DirectAddress=(uint32)MemPtr;
        ...
        bool IsInRange(const uint32 &address)
        {
            return( (address >= BaseAddress)
            && ( address <= EndAddress) );
        }
    }
};
```

```
bool Read( const uint32 &address, uint32
           &value)
{
    ASSERT( IsInRange( address ) );
    value = MemPtr[ ( address -
    BaseAddress ) / sizeof( uint32 ) ];
    return true;
}

bool Write( const uint32 &address,
            const uint32 value )
{
    ASSERT( IsInRange( address ) );
    MemPtr[ ( address - BaseAddress ) /
    sizeof( uint32 ) ] = value;
    return true;
}
```

# Simulated Bus (SIM\_BusMgr)

- SIMBUS\_Write is a static interface function that points to a static object SIM\_BusMgr

```
Class SIM_BusMgr : public SIM_Port
{
public:
    SIM_BusMgr();
    virtual ~SIM_BusMgr();
    bool MapPort(SIM_Port &newPort);
    bool Read(const uint32 &address, uint32
        &value)
    {
        SIM_Port *port =
        FindPortForAddress(address);
        if ( port != NULL )
        {
            return port->Read(address,value);
        }
        else
        {
            return false;
        }
    }
}

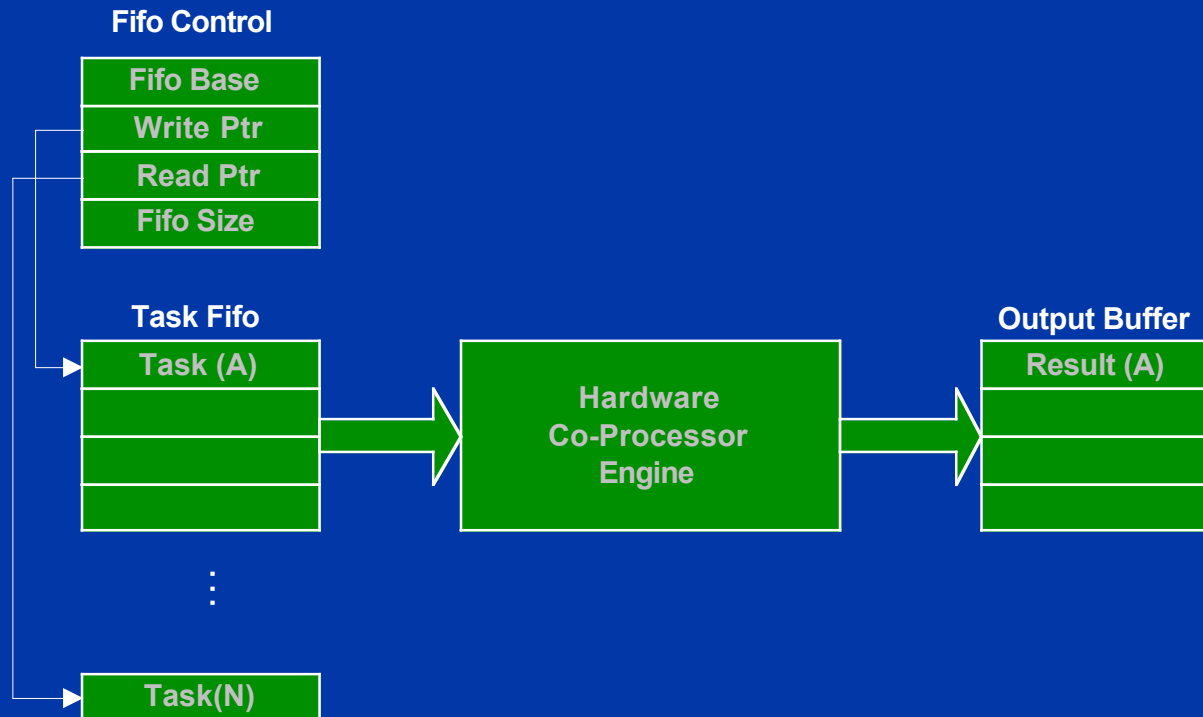
bool Write(const uint32 &address, const
    uint32 value )
{
    SIM_Port *port=
        FindPortForAddress(address);
    if ( port != NULL )
    {
        return port->Write(address,value);
    }
    else
    {
        return false;
    }
}

SIM_Port
    *MainPortList[SIM_BUS_MGR_MAX_PORTS];
unsigned int      MainPortNumElements;
```

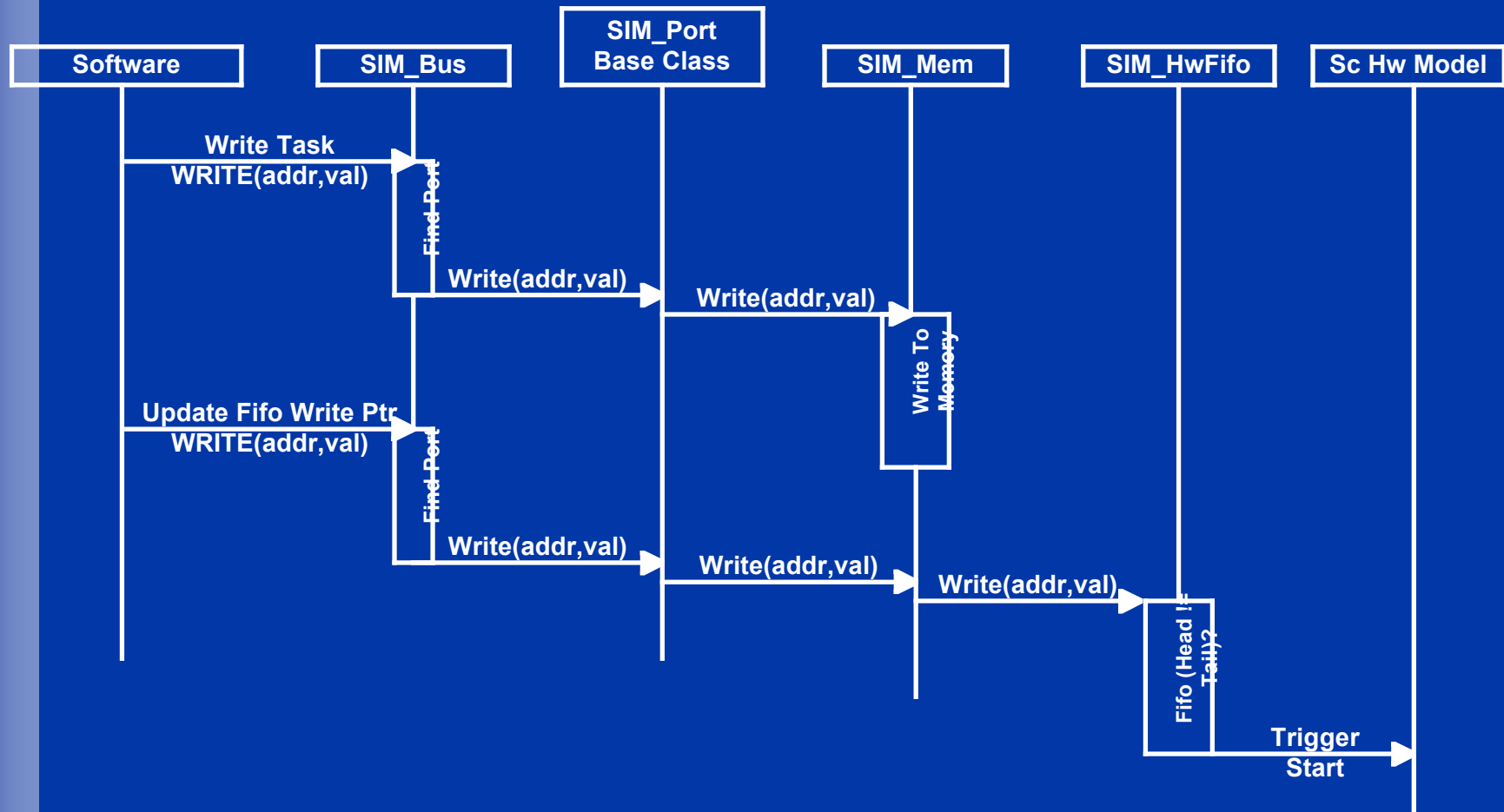
# Coprocessor Example

- A coprocessor is used to assist an embedded processor in executing a process.
- A software program prepares 3 tasks and writes them to HW task fifo for execution.
- HW coprocessor reads fifo, processes and outputs results to a buffer.
- Goal is to Co-Simulate HW and SW processing and show how we can gather information on total processing time.

# Coprocessor Hardware to Model

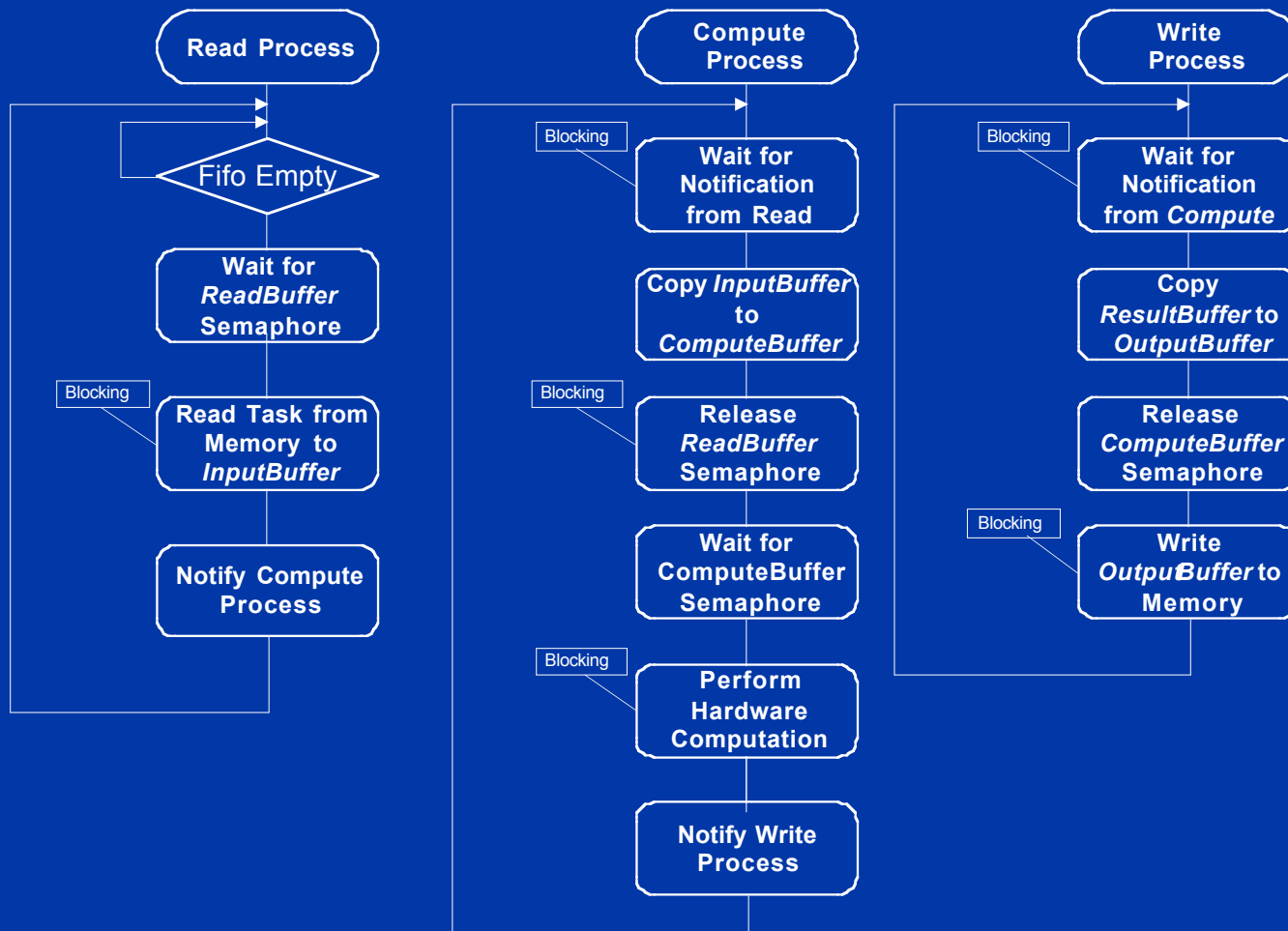


# Software Flow Writing to Hardware



# Hardware Processing

Hardware Processing  
with pipelining of Read, Compute  
and Write operations



# Hardware Model Code Example

```
SC_MODULE...
    SC_THREAD( ReadStageThread );
    SC_THREAD( ComputeStageThread );
    SC_THREAD( WriteStageThread );

    sensitive << ExternalWriteEvent;

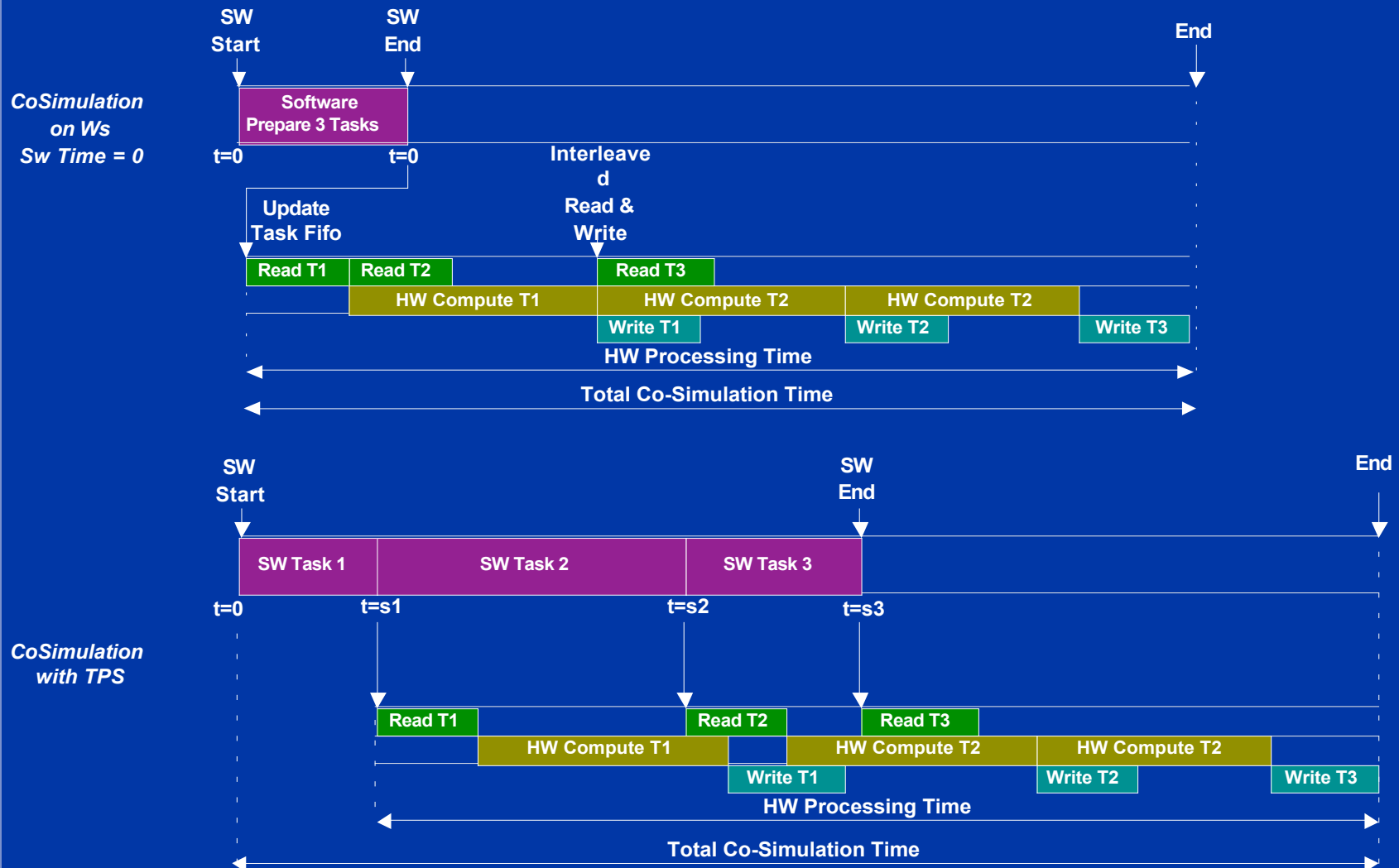
ReadStageThread()
{
    ComputeStageAvailableSemaphore.post();
    for( ;; )
    {
        wait(EncoderWriteEvent);
        while( !IsFifoEmpty() )
        {
            ReadStageTask =
            GetTaskFromFifoMemory();
            wait(readTimeUs, SC_US);

            ComputeStageAvailableSemaphore.wait();
            ComputeStageTask = ReadStageTask;
            StartComputeStageEvent.notify();
        }
    }
}

Write(const uint32 &address,
      const uint32 &data)
{
    if( address ==
        TASK_WRITE_PTR )
    {
        notify(ExternalWriteEvent);
    }
}
```

# Simulation

## Processing Timeline



# Conclusion

- Simulation on WS can go a long way to help software development, test interfaces, provide HW timing and functional verification.
- Simulation with TPS can provide software timing and help optimize Hw-Sw. It can also provide valuable performance metrics early in the development cycle and help make system design decisions.

## Tips:

- Define simulation goals and determine what model type best suits your goals (WS/TPS).
- Start with a simple bus-model to simulate hardware address space.
- Use building blocks for memory ports and re-use these blocks for active hardware blocks, they can expand into complex SystemC models.

# Acknowledgements

The Target Processor Simulator, also known as ISS (Instruction Set Simulator), used in this project was provided by:

Endeavor Intertech Corporation

[www.endeav.com](http://www.endeav.com)