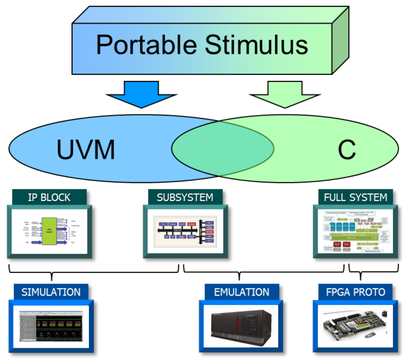
Managing and Automating Hw/Sw Tests from IP to SoC

Mentor® A Siemens Business

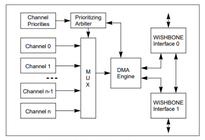
Matthew Ballance
matt_ballance@mentor.com

Firmware and Verification from IP to SoC



IP behavior often interacts closely with software
Beneficial to verify Hw/Sw interaction from IP to SoC

PSS Enables Test Intent Reuse



DMA Engine

```
component dma_c {
  buffer mem_seg_b {
    rand bit[31:0]   addr;
    rand bit[31:0]   size;
  }

  resource dma_channel_r { }

  pool dma_channel_r     channels[31];

  action dma_mem2mem_xfer_a {
    input mem_seg_b   src;
    output mem_seg_b  dst;
    lock dma_channel_r    channel;

    constraint size_match {
      src.size == dst.size;
    }

    // Target implementation left unspecified
  }
}
```

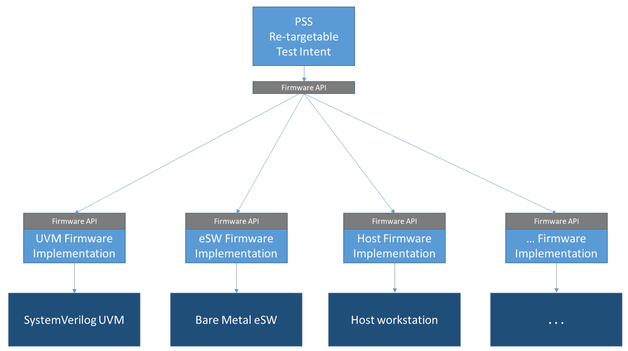
PSS DMA-Transfer Primitive

```
package dma_c_pkg {

  import void wb_dma_xfer(
    bit[31:0]                   channel,
    bit[31:0]                   src_addr,
    bit[31:0]                   dst_addr,
    bit[31:0]                   size
  );

  extend action dma_c::dma_mem2mem_xfer_a {
    exec body {
      wb_dma_xfer(
        channel.instance_id,
        src.addr,
        dst.addr,
        src.size
      );
    }
  }
}
```

PSS Mapping to Implementation

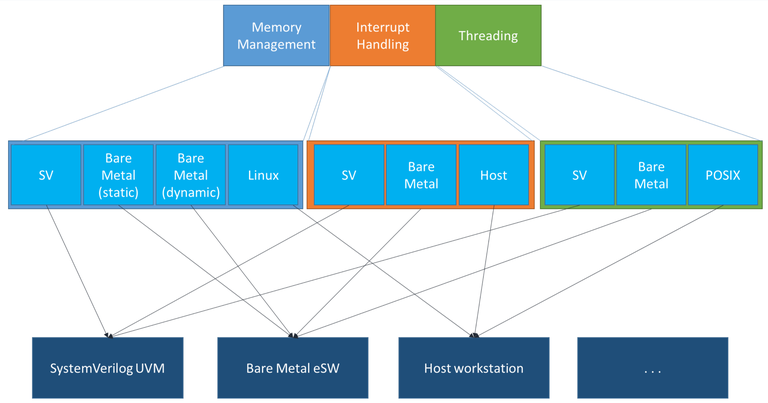
Portable Stimulus ensures reusable, consistent test intent
Boosts test-creation productivity

PSS: Reusable Test Intent, Platform-Specific Firmware



PSS enables test-intent portability
But, each target requires its own firmware implementation

Firmware Reuse with the Micro Executor API (UEX)



Provides services for firmware
Provides appropriate implementations for
- SystemVerilog / UVM
- Bare metal
- OS

Memory Services

```
void uex_iowrite8(uint8_t v, void *p);

void uex_iowrite16(uint16_t v, void *p);

void uex_iowrite32(uint32_t v, void *p);

void uex_iowrite64(uint64_t v, void *p);

uint8_t uex_ioread8(void *p);

uint16_t uex_ioread16(void *p);

uint32_t uex_ioread32(void *p);

uint64_t uex_ioread64(void *p);
```

```
void *uex_ioalloc(
                    uint32_t   sz,
                    uint32_t   align,
                    uint32_t   flags);

void uex_iofree(void *p);
```

```
void *uex_malloc(uint32_t sz);


void uex_free(void *p);
```

Abstracts access to target memory
Abstracts allocation/freeing of target and local memory

Threading Services

```
uex_thread_t uex_thread_create(
                    uex_thread_f                main_f,
                    void                *ud);

int uex_thread_join(uex_thread_t t);

void uex_yield(void);

uex_thread_t uex_thread_self(void);
```

```
void uex_mutex_init(uex_mutex_t *m);

void uex_mutex_lock(uex_mutex_t *m);

void uex_mutex_unlock(uex_mutex_t *m);

void uex_cond_init(uex_cond_t *c);

void uex_cond_wait(uex_cond_t *c, uex_mutex_t *m);

void uex_cond_signal(uex_cond_t *c);
```

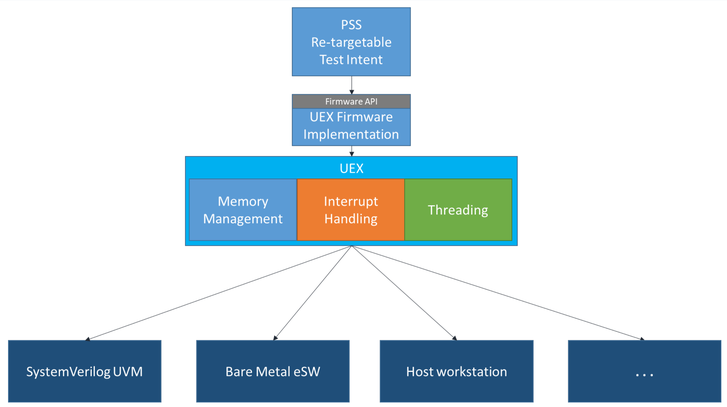
Provides thread creation and management services
Provides abstract synchronization primitives

Interrupt Services

```
typedef void (*uex_irq_f)(void *);

void uex_set_irq_handler(
                    uint32_t            irq,
                    uex_irq_f           f,
                    void                *ud);
```

Decouples firmware from a specific interrupt
Allows firmware to attach handles to specific interrupts
Abstracts from platform-specific interrupt support

PSS and UEX: Test Intent and Firmware Reuse



Same firmware runs on different platforms with UEX API
Reusable PSS test intent targets reusable firmware API
Same test intent and same firmware from IP to SoC

UEX Example – DMA Driver

```
void wb_dma_uex_drv_init(
  wb_dma_uex_drv_t              *drv,
  void                          *base) {
  uint32_t i;
  uex_device_t *dev;

  memset(drv, 0, sizeof(wb_dma_drv_t));
  drv->regs = (wb_dma_regs_t *)uex_ioremap(base, (5*4)+(8*4), 0);

  // Route enabled channels to INTA
  uex_iowrite32(&drv->regs->int_msk_a, 0xFFFFFFFF);
  uex_iowrite32(&drv->regs->int_src_a, 0xFFFFFFFF);

  for (i=0; i<31; i++) {
    // Enable for 256-byte chunks
    uex_iowrite32(&drv->regs->channels[i].size, ((256/4) << 16));

    // Configure address mask
    uex_iowrite32(&drv->regs->channels[i].src_msk, 0xFFFFFFFC);
    uex_iowrite32(&drv->regs->channels[i].dst_msk, 0xFFFFFFFC);
  }

  // Initialize thread-control fields
  for (i=0; i<31; i++) {
    uex_mutex_init(&drv->xfer_mutex[i]);
    uex_cond_init(&drv->xfer_cond[i]);
  }

  // Find the device and connect the interrupt based on the base
address
  dev = uex_find_device(base);
  uex_set_irq_handler(dev->irqs[0], &wb_dma_uex_irq, drv);
}
```

```
void wb_dma_uex_drv_single_xfer(
  wb_dma_uex_drv_t              *drv,
  uint32_t                      channel,
  uint32_t                      src,
  uint32_t                      inc_src,
  uint32_t                      dest,
  uint32_t                      inc_dst,
  uint32_t                      sz) {
  uint32_t csr, sz_v;
  uex_mutex_lock(&drv->xfer_mutex[channel]);

  // Program channel registers
  csr = uex_ioread32(&drv->regs->channels[channel].csr);

  csr |= (1 << 18); // interrupt on done
  csr |= (1 << 17); // interrupt on error
  csr =  (inc_src)?(csr | (1 << 4)):(csr & ~(1 << 4));
  csr =  (inc_dst)?(csr | (1 << 3)):(csr & ~(1 << 3));

  csr |= (1 << 2); // use interface 0 for source
  csr |= (1 << 1); // use interface 1 for destination

  csr |= (1 << 0); // enable channel

  // Setup source and destination addresses
  uex_iowrite32(&drv->regs->channels[channel].src, src);
  uex_iowrite32(&drv->regs->channels[channel].dst, dest);

  sz_v = uex_ioread32(&drv->regs->channels[channel].size);
  sz_v &= ~(0xFFF); // Clear tot_sz
  sz_v |= (sz & 0xFFF);
  uex_iowrite32(&drv->regs->channels[channel].size, sz_v);

  // Start the transfer
  uex_iowrite32(&drv->regs->channels[channel].csr, csr);

  drv->status[channel] = 1;

  // Wait for completion
  uex_cond_wait(&drv->xfer_cond[channel], &drv-
>xfer_mutex[channel]);

  uex_mutex_unlock(&drv->xfer_mutex[channel]);
}
```

Driver initialization maps memory and sets up mutexes
Transfer function accesses DMA registers via UEX
Transfer function uses mutex/cond to detect completion

Conclusion

UEX provides an abstract firmware-centric API
Provides implementations for simulation, bare-metal, OS
Enables modular, interoperable firmware

With PSS, enables Test Intent and firmware reuse IP to SoC

https://github.com/mballance/uex