

Advanced Computer Systems Architecture

Chip-Multiprocessors: Applications and Architectures

CSE 526M

Prof. Patrick Crowley

Plan for Today

- Questions
- Today's discussion

Objective

- Forced reading of ME assembly code
- Scratch ring mechanisms
- Preparing the MSF Receive Control CSR

Scratchpad Rings

- Purpose: share data between microengines, XScale and other units
- Characteristics
 - 16 (0 .. 15) scratch rings supported in HW
 - The first 11 have HW support for testing fullness
 - Once initialized, a ring supports get and put operations of between 1 and 8 longwords
- Steps
 - Initialize
 - Insert Data
 - Check for Fullness/Emptiness
 - Remove Data

Initializing/Creating a Ring

- Must configure three Ring-specific CSRs
 - SCRATCH_RING_BASE
 - SCRATCH_RING_HEAD
 - SCRATCH_RING_TAIL
- Programmer's Reference manual has details...

Initialization Macro

```
#macro scratch_ring_init(IN_RING_NUM, IN_RING_BASE, IN_RING_SIZE)
.begin

    .reg $ring_init_xfer $ring_head $ring_tail ring_init base_shift
    .sig ring_init_sig ring_head_sig ring_tail_sig

    immed[base_shift, IN_RING_BASE]
    alu_shf[ring_init, --, B, base_shift, <</**/RING_BASE_BITPOS]
    alu_shf_left[$ring_init_xfer, ring_init, OR, ((IN_RING_SIZE/128)-1), RING_SIZE_BITPOS]
    cap[write, $ring_init_xfer, SCRATCH_RING_BASE_/**/IN_RING_NUM], sig_done[ring_init_sig]

    immed32[$ring_head, 0]
    cap[write, $ring_head, SCRATCH_RING_HEAD_/**/IN_RING_NUM], sig_done[ring_head_sig]

    immed32[$ring_tail, 0]
    cap[write, $ring_tail, SCRATCH_RING_TAIL_/**/IN_RING_NUM], sig_done[ring_tail_sig]

    ctx_arb[ring_init_sig, ring_head_sig, ring_tail_sig]

.end
#endm
```

Inserting Data (put)

- Choose a ring
- Move data into up to 8 transfer registers
 - Use a xbuf's for this purpose
- 'put' data into ring

Put Macro

```
#macro scratch_ring_put_lw(IN_RING_NUM, in_data)
.begin
    .reg ring_addr
    .sig ring_sig
    // the following is a DECLARATION macro!, xbuf.uc.
    xbuf_alloc($ring_data, 1, write)

    immed32(ring_addr, (IN_RING_NUM<<2))

    move($ring_data[0], in_data)
    scratch[put, $ring_data[0], ring_addr, 0, 1], ctx_swap[ring_sig]

    xbuf_free($ring_data)
.end
#endm
```

Examining Fullness/Emptiness

- When is it safe to write/read?
- For 11 of the 16 rings, we can check ring status with a single instruction
 - `br_inp_state, br_!inp_state`
 - Note: these are *branch* instructions

Fullness Macro

```
#macro br_scratch_ring_full(IN_RING_NUM, IN_FULL_TARGET)
    br_inp_state[SCR_Ring/**/IN_RING_NUM/**/_Status, IN_FULL_TARGET]
    // You might need to use _Full rather than _Status (3.5 vs. 3.1)
    // SCR_Ring... understood by the decoder
#endm
```

Emptiness Macro

```
#macro br_scratch_ring_not_full(IN_RING_NUM, IN_FULL_TARGET)
    br_inp_state[SCR_Ring/**/IN_RING_NUM/**/_Status, IN_FULL_TARGET]
    // You might need to use _Full rather than _Status (3.5 vs. 3.1)
    // SCR_Ring... understood by the decoder
#endm
```

Removing Data (get)

- Choose a ring
- Prepare receive transfer registers (up to 8)
- 'get' the data from the ring

Get Macro

```
#macro scratch_ring_get_lw(IN_RING_NUM, out_data)
.begin
    .reg ring_addr
    .sig ring_sig

    xbuf_alloc($ring_data, 1, read)

    immed32(ring_addr, (IN_RING_NUM<<2))

    scratch[get, $ring_data[0], ring_addr, 0, 1], ctx_swap[ring_sig]
    move(out_data, $ring_data[0])

    xbuf_free($ring_data)
.end
#endm
```

Putting It All Together

- Example project

Put Performance

- 128 4-byte entries enqueued in 7565 cycles
 - Data size: 512B
 - Time: 7565 cycles @ 600 MHz (1.667 ns) = 12611 ns
 - Rate: 38.7 MB/s
- Bulk version
 - Time: 1743 cycles = 2906 ns
 - Rate: 168 MB/s

Preparing the MSF Receive Control CSR

3	3	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0													
1	0	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0													
RX_EN[3:0]				RESERVED								RX_MODE		RX_WIDTH		RX_MPHY_EN		RX_MPHY_POLL_MODE		TX_BUS_MODE		RESERVED		TX_BUS_WIDTH		RX_MPHY_POLL_MODE		RESERVED		CSIX_FREELIST		RESERVED				RBUF_ELEMENT_SIZE		RESERVED	

Manipulating CSRs

```
//-----  
// Bit positions within the MSF receive control CSR  
#define RX_MODE_BITPOS 22  
#define RX_WIDTH_BITPOS 20  
#define RX_MPHY_EN_BITPOS 19  
#define RX_MPHY_MODE_BITPOS 18  
#define RX_MPHY_POLL_MODE_BITPOS 17  
#define RX_MPHY_LEVEL2_BITPOS 14  
#define RX_RBUF_ELEMENT_SIZE 2  
#define RX_EN_BITPOS 28  
  
//If you change the next line you must change RX_RBUF_ELEMENT_SIZE below  
#define RBUF_ELEM_SIZE 128  
#define RX_CONTROL_VAL1 ((0 << RX_MODE_BITPOS) | \  
                        (00 << RX_WIDTH_BITPOS) | \  
                        (1 << RX_MPHY_EN_BITPOS) | \  
                        (0 << RX_MPHY_MODE_BITPOS) | \  
                        (0 << RX_MPHY_POLL_MODE_BITPOS) | \  
                        (0 << RX_MPHY_LEVEL2_BITPOS) | \  
                        (00 << RX_RBUF_ELEMENT_SIZE))  
  
//On the 2400, Rx_En must be set last  
#define RX_CONTROL_VAL2 ((RX_CONTROL_VAL1) | \  
                        (1 << RX_EN_BITPOS))
```

Writing CSRs

```
//----- Set the receive control CSR in the MSF.  
// 2XXX PRM p. 464  
immed32($rx_ctl_xfer, RX_CONTROL_VAL1)  
immed32(addr, MSF_RX_CONTROL_ADDR)  
msf[write, $rx_ctl_xfer, addr, 0, 1],  
      ctx_swap[msf_rx_ctl_sig]  
  
immed32($rx_ctl_xfer, RX_CONTROL_VAL2)  
immed32(addr, MSF_RX_CONTROL_ADDR)  
msf[write, $rx_ctl_xfer, addr, 0, 1],  
      ctx_swap[msf_rx_ctl_sig]
```

Assignment

- Commentary:
 - Choose one of the following (I'll post this project to the newsgroup)
 - How would you improve on the scratchpad ring put performance? Describe your design, implement it and report results.
 - Implement an analogous set of macros for SRAM rings. Describe the differences and performance and share your implementation in your commentary.
 - How does this compare to a dynamically-linked ring implementation on your desktop machine? Describe your experience in your commentary.