Design of Embedded DSP Processors, TSEA26

Date	2011-10-22
Room	TER3
Time	14:00-18:00
Course code	TSEA26
Exam code	TEN 1
Course name	Design of Embedded DSP Processors
Department	ISY, Department of EE
Number of questions	5
Number of pages (including this page)	7
Course responsible	Andreas Ehliar
Teacher visiting the exam room *Phone number during the exam time*	Andreas Ehliar
Visiting the exam room	Around 15 and 17
Course administrator	Ylva Jernling, 013-282648, ylva@isy.liu.se
Permitted equipment	None, besides an English dictionary
Grading	**Points** **Swedish grade** 41-50 5 31-40 4 21-30 3 0-20 U

- You can answer in English or Swedish. Don't write answers on the exam sheet.
- **When designing a hardware unit you should try minimize the amount of hardware used.** (Unless otherwise noted in the question.)
- The width of data buses and registers must be specified unless otherwise noted. Likewise, the alignment must be specified in all concatenations of signals or buses. When using a box such as *"SATURATE"* or *"ROUND"* in your schematic, you must (unless otherwise noted) describe the content of this box! (E.g. with RTL code). You can assume that all numbers are in two's complement representation unless otherwise noted in the question.
- In questions where you are supposed to write an assembler program based on pseudo code you are allowed to optimize the assembler program in various ways as long as the output of the assembler program is identical to the output from the pseudo code. (You can assume that the only output from a function is either the return value and/or any writes to memory performed in that function.)
- You can also (unless otherwise noted in the question) assume that hazards will not occur due to parts of the processor that you are not designing.

Good luck!

Problem 1: Address Generator Unit (13p)

You should create an address generator unit capable of supporting the following function:

```
// Inputs/outputs: xoffs1 is passed in r0, yoffs1 in r1
//                 xoffs2 is passed in r2, yoffs2 in r3
//                 img and ref are arrays containing 16-bit data
//
// Note: You don't need to worry about buffer overflows, it is up to
//       the caller of blit() to ensure that xoffs1-yoffs2 contains
//       reasonable values.
function blit(xoffs1, yoffs1,xoffs2,yoffs2)
    for(y=0; y < 8; y++)
        for(x=0; x < 8; x++)
            img[x+xoffs1+(y+yoffs1)*160] = ref[x+xoffs2+(y+yoffs2)*160]
        endfor
    endfor
endfunction

// Inputs: s is in r0, t is in r1,  boundaryflag is in r2
//         abs(s) and abs(t) are guaranteed to be less than 32000
function getit(s,t,boundaryflag)
    tmp = getsingle(s,t,boundaryflag)
    tmp = tmp + getsingle(s,t+1,boundaryflag
    tmp = tmp + getsingle(s+1,t,boundaryflag)
    tmp = tmp + getsingle(s+1,t+1,boundaryflag)
    return tmp // Return value in r0 or acr0 (your choice)
endfunction

function getsingle(s,t, boundaryflag)
    if boundaryflag == 1 then
        if s < 0 then
            s = 0
        elseif s > 63 then
            s = 63
        endif
        if t < 0 then
            t = 0
        elseif t > 63 then
            t = 63;
        endif
    else
        s = s & 63;
        t = t & 63;
    endif
    return tmem[s+t*64]; // tmem is a 16-bit wide array in DM0
endfunction
```

Constraints:

- **The blit() function should execute in at most 240 clock cycles!**
- **The getit() function should execute in at most 12 clock cycles**
- `getsingle()` will only be called by the `getit()` function. *Hint: Use inlining to avoid the cost of a function call!*
- The processor is a typical 16 bit DSP processor (that is, the registers are 16 bits wide and the address to DM0 is supposed to be 16 bits wide.
- You can assume that the processor has all instructions you need to implement these two programs within the given constraints. (Within reason.) For example, you can assume that the processor can handle at least two levels of nested repeat instructions. You can also assume that both the ALU and the MAC unit are connected to DM0.
- You can assume that the processor has full forwarding/bypass
- The values and algorithms in the functions listed above have been carefully selected so that no hardware multipliers should be necessary in the AGU.

a) List the addressing modes required to run these programs under the given constraints and translate the functions into assembler. If you don't answer exercise b below you need to carefully describe what each addressing mode does as well. (6p)

b) Draw a schematic of your AGU and a control table with all AGU operations that are needed to support the programs that you translated in part a. (7p)

Problem 2: Arithmetic Unit (8p)

Draw a schematic and create a control table for an ALU capable of the following operations:

- **OP1:** `A+B`
- **OP2:** `A-B`
- **OP3:** `SAT(A+B)`
- **OP4:** `SAT(A-B)`
- **OP5:** `(A+B)/2`
- **OP6:** `SAT(ABS(A)+ABS(B))`
- **OP7:** `SAT(ABS(A-B))`

The inputs (A and B) are 16 bit wide. The result should be 16 bit wide.

Problem 3: Program flow control (PFC) unit (10p)

You should draw a schematic and control table for a PFC unit capable of the following operations:

- **OP1:** Reset (PC = 0x0 and the hardware stack is cleared)
- **OP2:** NOP
- **OP3:** PC = PC + 1
- **OP4:** PC = PC + SIGNEXT(IMM[11:0])
- **OP5:** if(flag == 1) PC = PC + SIGNEXT(IMM[11:0]) else PC = PC + 1
- **OP6:** if(flag == 0) PC = PC + SIGNEXT(IMM[11:0]) else PC = PC + 1
- **OP7:** PC = OpA
- **OP8:** Push(PC); PC = PC + SIGNEXT(IMM[11:0])
- **OP9:** Push(PC); PC = OpA
- **OP10:** PC = Pop()+1

Constraints:

- Your hardware stack (used for OP8-OP10) should contain two entries.
- You don't need to worry about pipeline issues in this exercise (you can assume that the instruction decoder has already taken care of delay slot handling, etc)

Required inputs and outputs:[1]

- (Input) Clock signal (of course)
- (Input) Control signals from instruction decoder
- (Input) IMM[11:0] (from instruction word)
- (Input) OpA[15:0] (from register file)
- (Output) TO_PM[15:0] - address to program memory
- (Output) Stack_Error - Set to one when the hardware stack is full and OP8 or OP9 is executed. Alternatively it should be set to one when the hardware stack is empty and OP10 is executed. Otherwise it should be 0.

[1]Errata: There should be a one bit flag input here as well. (The same flag as used in OP5 and OP6.)

Problem 4: General knowledge (7p)

a) Consider the following pseudo code and list of assembler instructions:

```
// Pseudo code
tmp = -32768
for(i=0; i < 90; i = i + 1)
    tmp = max(tmp, a[i])
endfor
```

```
alternative1:
    set  ar0,a_ptr
    set r3,-32768
    repeat 90, endloop
    ld r0,DM0[ar0++]
    max r3,r0,r3
endloop:
    ret
```

```
alternative2:
    set  ar0,a_ptr
    set r3,-32768
    set r4,-32768
    repeat 45, endloop
    ld r0,DM0[ar0++]
    ld r1,DM0[ar0++]
    max r3,r0,r3
    max r4,r1,r4
endloop:
    max r3,r4,r3
    ret
```

Explain, using at most five sentences, why alternative 2 may be preferable to use over alternative 1 in an application specific processor even though slightly more instructions need to be executed in alternative 2. (2p)

b) Explain, using at most five sentences, the concept of hardware multiplexing. Give one example of hardware multiplexing in an ALU. (2p)

c) Bit-reversed addressing is commonly included in DSP processors. Name one commonly encountered DSP algorithm which is simplified by having such an addressing mode (1p)

d) Draw a very simple processor pipeline containing the following parts: Program Counter, Program memory, Instruction decoder, Register file, and Writeback stage (you don't need to draw the contents of these parts, e.g., drawing a box with the letters "PC" inside is enough for the program counter).

Use this figure to explain (using at most five sentences) the concept of delay slots (2p)

Problem 5: MAC unit (12p)

You should create a MAC unit which can support the functions shown on the next page under the given constraints:

```
// ptr0 is passed in ar0, ptr1 in ar1
function filt1(ptr0, ptr1)
    tmp = 0
    repeat(32)
        // Note: Signed integer multiplication!
        tmp = tmp + DM0[ptr0++]*DM1[ptr1++]
    endrepeat

    if(tmp > 0x7fff)
        tmp = 0x7fff
    elseif(tmp < -0x8000)
        tmp = -0x8000
    endif
    return tmp    // The return value should be in r0
endfunction

// ptr0 is passed in r0, ptr1 in r1 and ptr2 in r2
function filt2(ptr0, ptr1, ptr2)
  tmp1 = 0
  tmp2 = 0
  repeat(20)
    // Note: Signed integer multiplications!
    tmp1 = tmp1 + DM0[ptr0++]*DM1[ptr1]   // Note: ptr1 should only
    tmp2 = tmp2 + DM0[ptr0++]*DM1[ptr1++] // be incremented once here!
  endrepeat

  tmp1 = tmp1 * 2
  tmp2 = tmp2 * 2
  tmp1 = tmp1 + 0x8000
  tmp2 = tmp2 + 0x8000
  tmp1 = tmp1 >> 16 // arithmetic right shift by 16
  tmp2 = tmp2 >> 16 // arithmetic right shift by 16

  if(tmp1 > 0x7fff)
    tmp1 = 0x7fff
  elseif(tmp1 < -0x8000)
    tmp1 = -0x8000
  endif
  if(tmp2 > 0x7fff)
    tmp2 = 0x7fff
  elseif(tmp2 < -0x8000)
    tmp2 = -0x8000
  endif

  DM0[ptr2+0] = tmp1
  DM0[ptr2+1] = tmp2
endfunction
```

Allowed inputs to the MAC unit:

- Clock signal
- Control signals from the instruction decoder
- DM0[15:0] - Data from DM0
- DM1[15:0] - Data from DM1

Allowed output:

- TO_RF[15:0] - Sent to the writeback port on the register file

Other constraints:

- **The filt1() function should execute in at most 40 cycles, not counting the ret instruction.**
- **The filt2() function should execute in at most 58 cycles, not counting the ret instruction.**
- The accumulator should be 40 bits wide
- You can assume that the processor has all instructions that are required to implement these functions. (Basically all instructions you would find in a simple DSP processor like Senior.) For example, the processor has a repeat instruction and the AGU:s have the appropriate addressing modes, etc.

a) Design an instruction set for your MAC unit that allows you to implement the functions shown above under the given constraints. Translate the filt1() and filt2() function into assembler. If you don't answer part b of this question you should also clearly explain what each instruction does (6p)

b) Draw a schematic and a control table of your MAC unit. (6p)