SOC LAB

LAB D Report

Group 13

2024/01/04

- Sdram controller interface

```
sdram_controller user_sdram_controller (
    .clk(clk),
    .rst(rst),

    .sdram_cle(sdram_cle),
    .sdram_cs(sdram_cs),
    .sdram_cas(sdram_cas),
    .sdram_ras(sdram_ras),
    .sdram_we(sdram_we),
    .sdram_dqm(sdram_dqm),
    .sdram_ba(sdram_ba),
    .sdram_a(sdram_a),
    .sdram_dqi(d2c_data),
    .sdram_dqo(c2d_data),

    .user_addr(ctrl_addr),
    .rw(wbs_we_i),
    .data_in(wbs_dat_i),
    .data_out(wbs_dat_o),
    .busy(ctrl_busy),
    .in_valid(ctrl_in_valid),
    .out_valid(ctrl_out_valid)
);
```

SDRAM 主要的四個控制訊號 CS、RAS、CAS、WE 可 decode 成以下七個狀態:

```
// Commands Decode
wire      Active_enable    = ~Cs_n & ~Ras_n &  Cas_n &  We_n;
wire      Aref_enable      = ~Cs_n & ~Ras_n & ~Cas_n &  We_n;
wire      Burst_term       = ~Cs_n &  Ras_n &  Cas_n & ~We_n;
wire      Mode_reg_enable  = ~Cs_n & ~Ras_n & ~Cas_n & ~We_n;
wire      Prech_enable     = ~Cs_n & ~Ras_n &  Cas_n & ~We_n;
wire      Read_enable      = ~Cs_n &  Ras_n & ~Cas_n &  We_n;
wire      Write_enable     = ~Cs_n &  Ras_n & ~Cas_n & ~We_n;
```

- Firmware matmul executed in SDRAM

將.hex 檔換成 matmux.hex

```
    spiflash #(
        //.FILENAME("counter_la.hex")
        .FILENAME("matmux.hex")
    ) spiflash (
```

模擬結果:

```
kai@kai-VirtualBox:~/caravel-soc_fpga-lab/lab-sdram/testbench/counter_la$ source
 run_sim
Reading matmux.hex
matmux.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0050
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0016
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x001c
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0022
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value p
assed, 0x0028
LA Test 2 passed
```

- load address/data in two different bank

原先 firmware 中的 data 會存在 dff，以下為 counter_la_mm.out 所呈現的結果。

```
00000000 <A>:
   0:   0000                    .2byte  0x0
   2:   0000                    .2byte  0x0
   4:   0001                    .2byte  0x1
   6:   0000                    .2byte  0x0
   8:   0002                    .2byte  0x2
   a:   0000                    .2byte  0x0
   c:   00000003                lb      zero,0(zero) # 0 <A>
  10:   0001                    .2byte  0x1
  12:   0000                    .2byte  0x0
  14:   0000                    .2byte  0x0
  16:   0000                    .2byte  0x0
  18:   0002                    .2byte  0x2
  1a:   0000                    .2byte  0x0
  1c:   00000003                lb      zero,0(zero) # 0 <A>
  20:   00000003                lb      zero,0(zero) # 0 <A>
  24:   0001                    .2byte  0x1
  26:   0000                    .2byte  0x0
  28:   0000                    .2byte  0x0
  2a:   0000                    .2byte  0x0
  2c:   0002                    .2byte  0x2
  2e:   0000                    .2byte  0x0
  30:   00000003                lb      zero,0(zero) # 0 <A>
  34:   0002                    .2byte  0x2
  36:   0000                    .2byte  0x0
  38:   0001                    .2byte  0x1
  3a:   0000                    .2byte  0x0
  3c:   0000                    .2byte  0x0
        ...

00000040 <B>:
  40:   0001                    .2byte  0x1
  42:   0000                    .2byte  0x0
  44:   0002                    .2byte  0x2
  46:   0000                    .2byte  0x0
  48:   00000003                lb      zero,0(zero) # 0 <A>
  4c:   0004                    .2byte  0x4
  4e:   0000                    .2byte  0x0
  50:   0005                    .2byte  0x5
  52:   0000                    .2byte  0x0
  54:   0006                    .2byte  0x6
  56:   0000                    .2byte  0x0
  58:   00000007                .4byte  0x7
  5c:   0008                    .2byte  0x8
  5e:   0000                    .2byte  0x0
  60:   0009                    .2byte  0x9
  62:   0000                    .2byte  0x0
  64:   000a                    .2byte  0xa
  66:   0000                    .2byte  0x0
  68:   0000000b                .4byte  0xb
  6c:   000c                    .2byte  0xc
  6e:   0000                    .2byte  0x0
  70:   000d                    .2byte  0xd
  72:   0000                    .2byte  0x0
  74:   000e                    .2byte  0xe
  76:   0000                    .2byte  0x0
  78:   0000000f                fence   unknown,unknown
  7c:   0010                    .2byte  0x10
```

在 section.lds 文件中調整 address map 使 execute code 存於 bank0、bank1(因 matmux 的 code 較長，一個 bank 裝不下)。而矩陣 A、B 的 data 存於 bank2。Address 的位置如下圖:

```
MEMORY {
        vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
        dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
        dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
        flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
        mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
        mprjram : ORIGIN = 0x38000000, LENGTH = 0x00000200
        mm_data : ORIGIN = 0x38000200, LENGTH = 0x00000200
        hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
        csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

將.data 與.bss 的資料由原先存放的 dff 改至我們定義出的 mm_data。

```
    .data :
    {
            . = ALIGN(8);
            _fdata = .;
            *(.data .data.* .gnu.linkonce.d.*)
            *(.data1)
            _gp = ALIGN(16);
            *(.sdata .sdata.* .gnu.linkonce.s.*)
            . = ALIGN(8);
            _edata = .;
    } > mm_data AT > flash

    .bss :
    {
            . = ALIGN(8);
            _fbss = .;
            *(.dynsbss)
            *(.sbss .sbss.* .gnu.linkonce.sb.*)
            *(.scommon)
            *(.dynbss)
            *(.bss .bss.* .gnu.linkonce.b.*)
            *(COMMON)
            . = ALIGN(8);
            _ebss = .;
            _end = .;
    } > mm_data AT > flash
```

將 data 存入 user memory:

```
38000200 <A>:
38000200:       0000                    .2byte  0x0
38000202:       0000                    .2byte  0x0
38000204:       0001                    .2byte  0x1
38000206:       0000                    .2byte  0x0
38000208:       0002                    .2byte  0x2
3800020a:       0000                    .2byte  0x0
3800020c:       00000003                lb      zero,0(zero) # 0 <__DYNAMIC>
38000210:       0001                    .2byte  0x1
38000212:       0000                    .2byte  0x0
38000214:       0000                    .2byte  0x0
38000216:       0000                    .2byte  0x0
38000218:       0002                    .2byte  0x2
3800021a:       0000                    .2byte  0x0
3800021c:       00000003                lb      zero,0(zero) # 0 <__DYNAMIC>
38000220:       00000003                lb      zero,0(zero) # 0 <__DYNAMIC>
38000224:       0001                    .2byte  0x1
38000226:       0000                    .2byte  0x0
38000228:       0000                    .2byte  0x0
3800022a:       0000                    .2byte  0x0
3800022c:       0002                    .2byte  0x2
3800022e:       0000                    .2byte  0x0
38000230:       00000003                lb      zero,0(zero) # 0 <__DYNAMIC>
38000234:       0002                    .2byte  0x2
38000236:       0000                    .2byte  0x0
38000238:       0001                    .2byte  0x1
3800023a:       0000                    .2byte  0x0
3800023c:       0000                    .2byte  0x0
        ...

38000240 <B>:
38000240:       0001                    .2byte  0x1
38000242:       0000                    .2byte  0x0
38000244:       0002                    .2byte  0x2
38000246:       0000                    .2byte  0x0
38000248:       00000003                lb      zero,0(zero) # 0 <__DYNAMIC>
3800024c:       0004                    .2byte  0x4
3800024e:       0000                    .2byte  0x0
38000250:       0005                    .2byte  0x5
38000252:       0000                    .2byte  0x0
38000254:       0006                    .2byte  0x6
38000256:       0000                    .2byte  0x0
38000258:       00000007                .4byte  0x7
3800025c:       0008                    .2byte  0x8
3800025e:       0000                    .2byte  0x0
38000260:       0009                    .2byte  0x9
38000262:       0000                    .2byte  0x0
38000264:       000a                    .2byte  0xa
38000266:       0000                    .2byte  0x0
38000268:       0000000b                .4byte  0xb
3800026c:       000c                    .2byte  0xc
3800026e:       0000                    .2byte  0x0
38000270:       000d                    .2byte  0xd
38000272:       0000                    .2byte  0x0
38000274:       000e                    .2byte  0xe
38000276:       0000                    .2byte  0x0
38000278:       0000000f                fence   unknown,unknown
3800027c:       0010                    .2byte  0x10
```

- Waveform

```
int A[SIZE*SIZE] = {0, 1, 2, 3,
                    1, 0, 2, 3,
                    3, 1, 0, 2,
                    3, 2, 1, 0,
};
int B[SIZE*SIZE] = {1, 2, 3, 4,
                    5, 6, 7, 8,
                    9, 10, 11, 12,
                    13, 14, 15, 16,
```

寫入矩陣 data

矩陣 data 被分配在 bank 2。