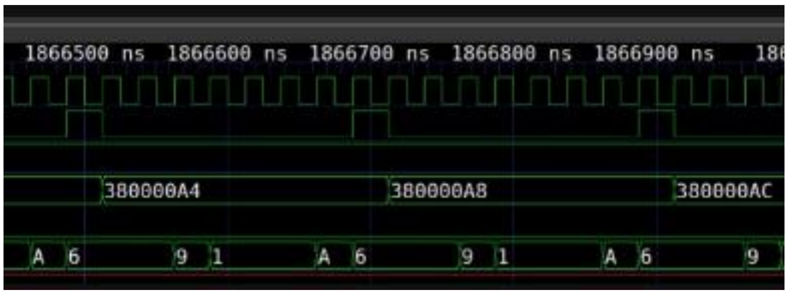
SOC lab D

310511082 王品棠 311510060 李秉翰 311512030 曾子鈞

- The SDRAM controller design, SDRAM bus protocol
  1. In IDLE state, controller first check wether the sdram need to be refreshed
  2. If not , it start trying to reading or writing .
  3. Beforing read or write, check if its on page. If not, start reading or writing or precharge.if it is, it go into ACTIVATE state.
  4. System go into PRECHARGE state at the begining we use new activate row.
- Introduce the prefetch scheme

  We didnt use prefetch buffer, but keep reading next address data without reading wishbone addr (controller do this in IDLE state)until the {7'b0, addr_q[7:2]} + prefetch_counter != current data address from wishbone

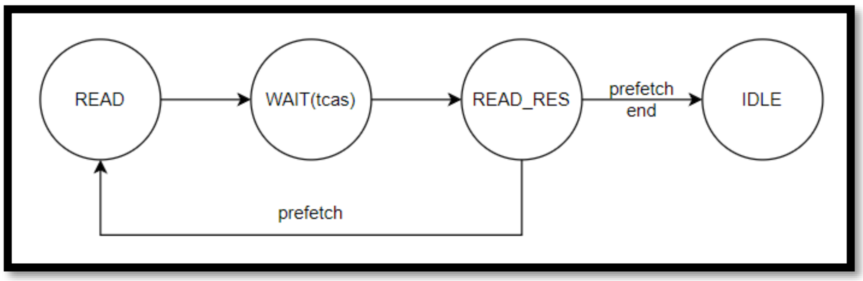
  (a) Original data reading waveform: 8T per data reading

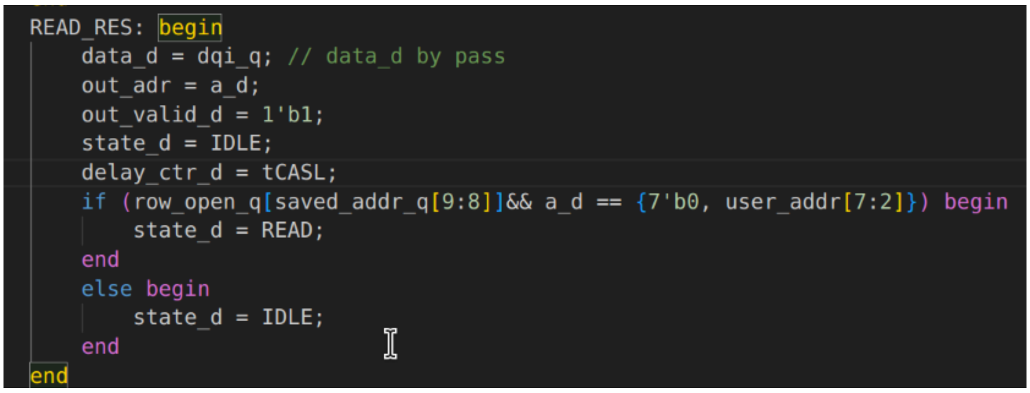


The original controller design takes 3T in IDLE state(6) per data reading.
So, we modified the state machine to keep the state from being IDLE between per data reading.

(b) SDRAM controller state machine modified

$$\begin{tikzcd}
\text{READ} \arrow[r] & \text{WAIT(tcas)} \arrow[r] & \text{READ\_RES} \arrow[r, "\text{prefetch end}"] \arrow[ll, bend left, "\text{prefetch}"] & \text{IDLE}
\end{tikzcd}$$

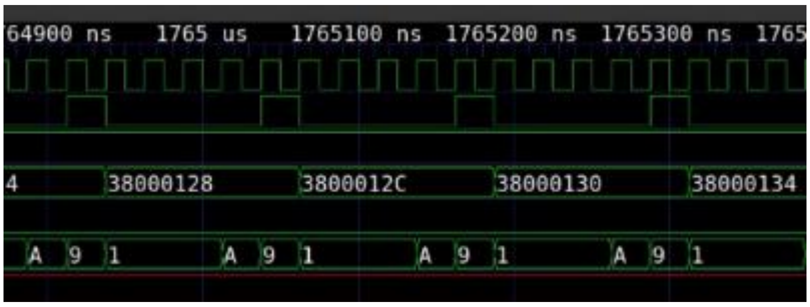




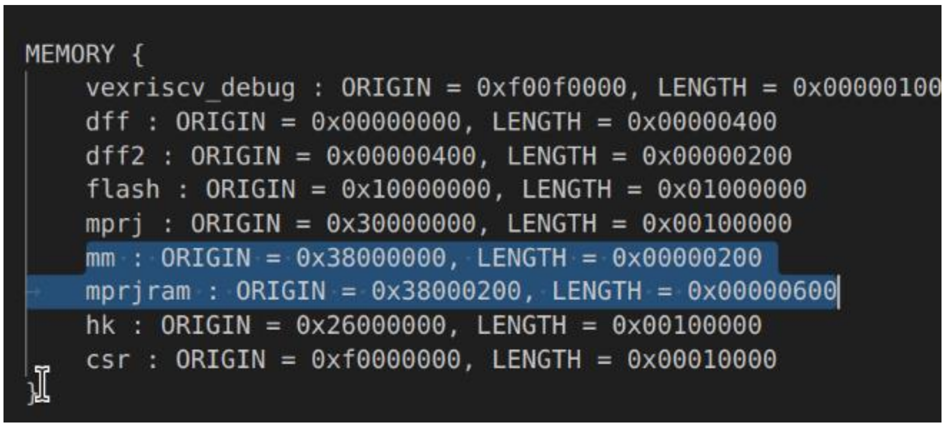
```
READ_RES: begin
    data_d = dqi_q; // data_d by pass
    out_adr = a_d;
    out_valid_d = 1'b1;
    state_d = IDLE;
    delay_ctr_d = tCASL;
    if (row_open_q[saved_addr_q[9:8]]&& a_d == {7'b0, user_addr[7:2]}) begin
        state_d = READ;
    end
    else begin
        state_d = IDLE;
    end
end
```

If current prefetch data reading is on page and match the address from wishbone, We'll keep doing prefetch. If not, it go back to IDLE state.

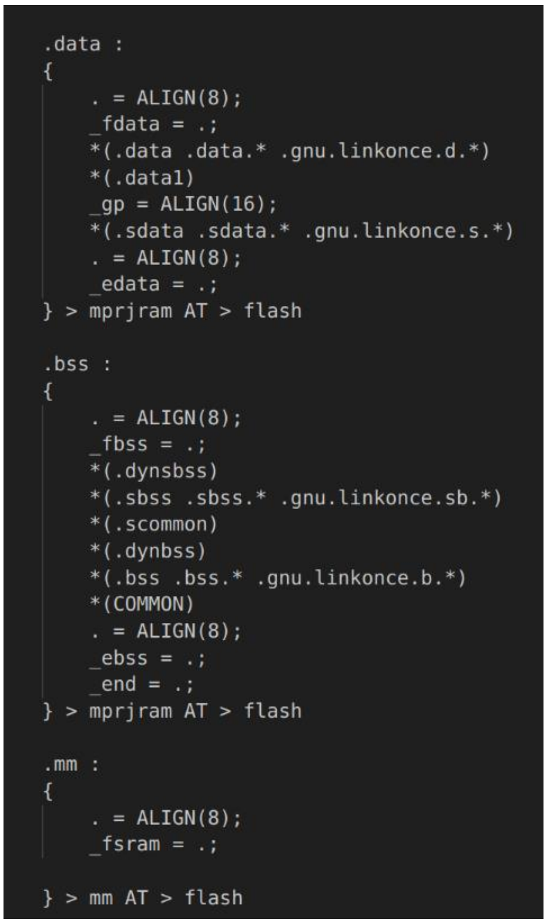
© New data reading waveform: 5T per data reading



- Introduce the bank interleave for code and data
  When cpu excute firmware code, it read instruction code in order until some parts it read data.
  For enhancing system with reading prefetch, we store code and data in different banks
- Introduce how to modify the linker to load address/data in two different bank
  Because we decode bank address in addr[9:8],we can devide bank into 4 part (0-100, 100-200,200-300, 300-400 address)
  (a) Linker modification: section mm for instruction code; section mprjram for data

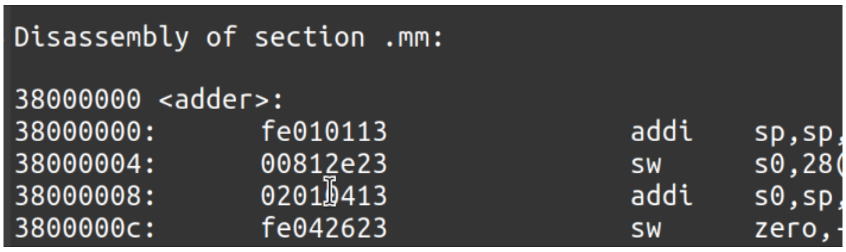


```
MEMORY {
    vexriscv_debug : ORIGIN = 0xf00f0000, LENGTH = 0x00000100
    dff : ORIGIN = 0x00000000, LENGTH = 0x00000400
    dff2 : ORIGIN = 0x00000400, LENGTH = 0x00000200
    flash : ORIGIN = 0x10000000, LENGTH = 0x01000000
    mprj : ORIGIN = 0x30000000, LENGTH = 0x00100000
    mm : ORIGIN = 0x38000000, LENGTH = 0x00000200
    mprjram : ORIGIN = 0x38000200, LENGTH = 0x00000600
    hk : ORIGIN = 0x26000000, LENGTH = 0x00100000
    csr : ORIGIN = 0xf0000000, LENGTH = 0x00010000
}
```

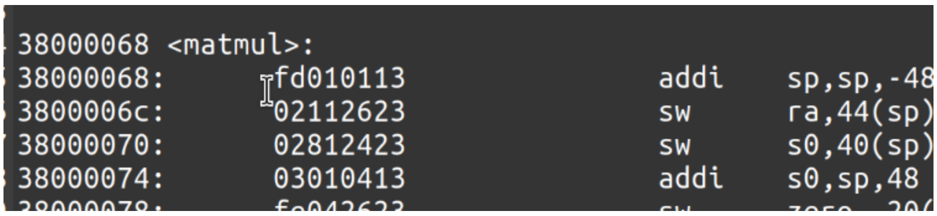


```
.data :
{
    . = ALIGN(8);
    _fdata = .;
    *(.data .data.* .gnu.linkonce.d.*)
    *(.data1)
    _gp = ALIGN(16);
    *(.sdata .sdata.* .gnu.linkonce.s.*)
    . = ALIGN(8);
    _edata = .;
} > mprjram AT > flash

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

.mm :
{
    . = ALIGN(8);
    _fsram = .;

} > mm AT > flash
```

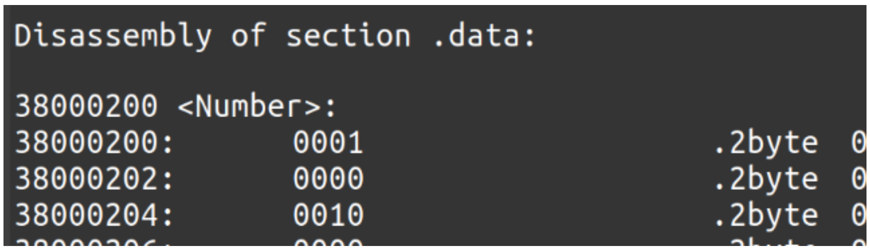
(b) .out



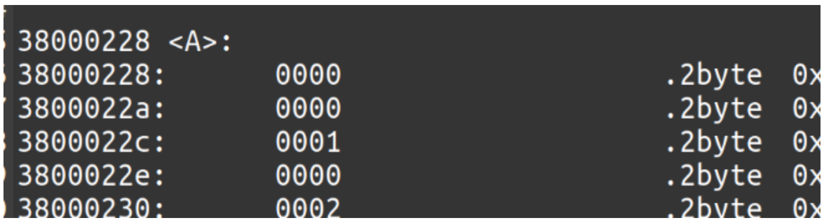
```
Disassembly of section .mm:

38000000 <adder>:
38000000:       fe010113                addi    sp,sp,
38000004:       00812e23                sw      s0,28(
38000008:       02010413                addi    s0,sp,
3800000c:       fe042623                sw      zero,-
```



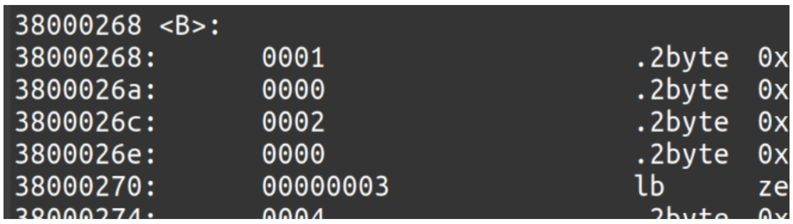
```
38000068 <matmul>:
38000068:       fd010113                addi    sp,sp,-48
3800006c:       02112623                sw      ra,44(sp)
38000070:       02812423                sw      s0,40(sp)
38000074:       03010413                addi    s0,sp,48
```



```
Disassembly of section .data:

38000200 <Number>:
38000200:       0001                    .2byte  0
38000202:       0000                    .2byte  0
38000204:       0010                    .2byte  0
```

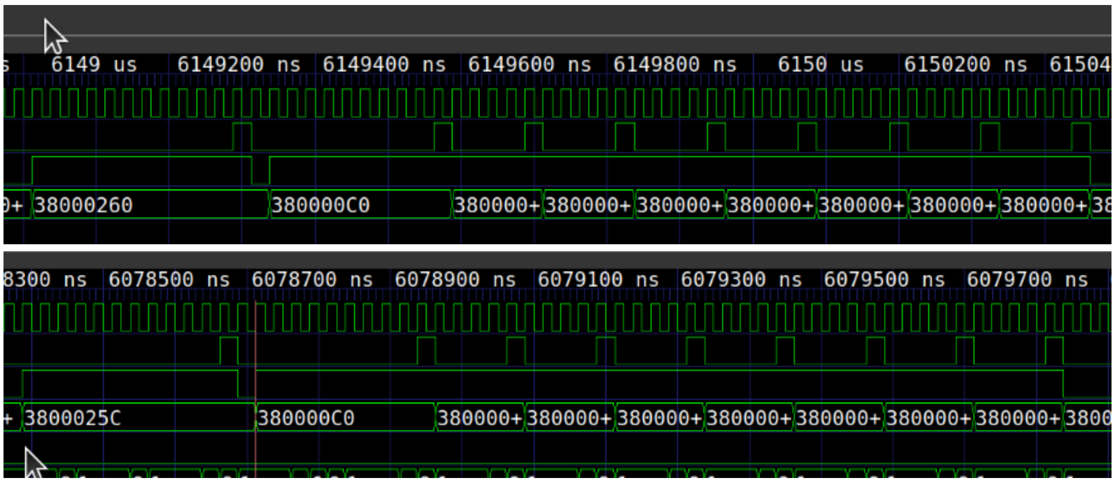


```
38000228 <A>:
38000228:       0000                    .2byte  0x
3800022a:       0000                    .2byte  0x
3800022c:       0001                    .2byte  0x
3800022e:       0000                    .2byte  0x
38000230:       0002                    .2byte  0x
```



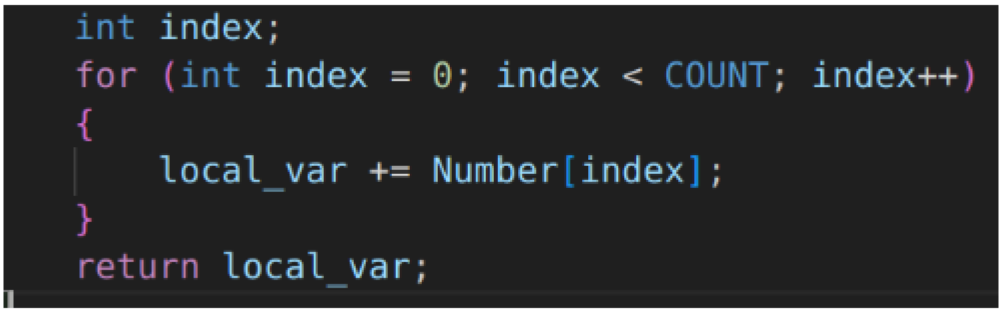
```
38000268 <B>:
38000268:       0001                    .2byte  0x
3800026a:       0000                    .2byte  0x
3800026c:       0002                    .2byte  0x
3800026e:       0000                    .2byte  0x
38000270:       00000003                lb      ze
38000274:       0004                    .2byte  0x
```

(c) Adder Waveform



In adder waveform, it read instruction code sequencely with prefetch, and read data frequently in the for loop below.

(c) Adder instruction code for loop



```
    int index;
    for (int index = 0; index < COUNT; index++)
    {
        local_var += Number[index];
    }
    return local_var;
```

- In lab d, we do the prefetch with burst length 1(singer data access), which is unefficiency.
  So the final project we want to add burst to this lab project.