DIT-FFT 至简设计实现法

1、 DIT-FFT 算法的基本原理

有限长序列x_n的 N 点 DFT 定义为：$\mathrm{X(k)} = \sum_{n=0}^{N-1} x(n) W_N^{Kn}$, 式中$W_N = e^{-j\frac{2\pi}{N}}$。DFT 在实际应用中很重要，但是如果直接按 DFT 变换进行计算，当序列长度 N 很大时，计算量会非常大，所需时间也很长，因此常用的是 DFT 的一种快速计算算法，简称 FFT。

最常用的 FFT 算法是基于时间抽取的基 2-FFT 算法和基于频率抽取的基 2-FFT 算法，这种算法的特点在于 FFT 会把一次大的 DFT 分割成几个小的 DFT，这样递归式地细分下去，例如有 8 个采样点的 FFT，首先会把最外层的 8 点运算分成两个 4 点 FFT 的奇偶组合，第二层 FFT 又分成四个两点 FFT 的奇偶组合，并且由此计算出的频谱中很有趣的一点在于对于实数输出的数组，后面一半和前面一半正好对称相同，对于虚数输出的数组，后面一半是前面数组对称后乘上负 1，因此，我们只需要算出 FFT 的一半即可求出全部。

本设计讨论的是基于**至简设计法**实现按时间抽选的基 2-FFT 算法（即 DIF-FFT）实现过程，支持 N 由 8 到 1024。

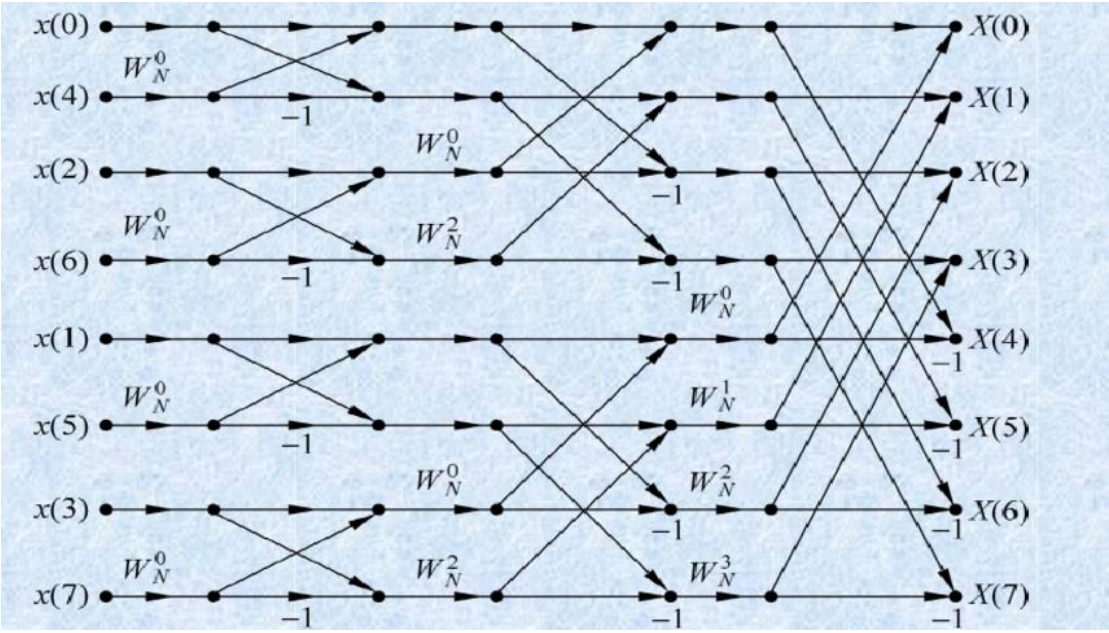


图 1 按时间抽取的基 2-FFT 算法蝶形运算流图（N=8）

2、 蝶形运算至简实现过程

2、1 模块划分

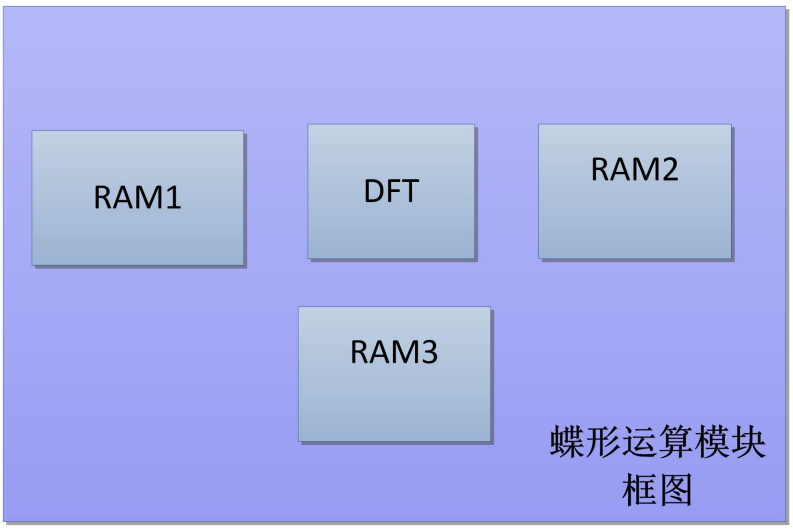


图 2 蝶形运算模块框图

本模块包括三个 RAM 模块（RAM1，RAM2，RAM3）与一个 DFT 模块，各模块功能如下：

1） RAM1 模块：在开始进行蝶形运算前，全部采样点（如图 1 所示的 x(0)、x(4)、x(2)、x(6)、x(1)、x(5)、x(3)、x(7)）已经按照倒位序二进制的地址依次存储在 RAM1 模块中，即地址 0 保存了采样点 x(0)，地址 1 保存了采样点 x(4)。选用双端口 RAM1 可以同时对两点采样数据(如图 1 的 x（0)、x(4))进行读、写操作。

2） RAM2 模块：RAM2 模块也是采用双端口输入输出，可同时对两点数据进行读、写操作。

3） DFT 模块：DFT 模块用于对 RAM1、RAM2 输出的两点采样数据（如图 1 的 x（0)、x(4)）进行蝶形运算，它将运算结果输出至 RAM1、RAM2 模块进行保存。

4） RAM3 模块：RAM3 模块是单输出模块，理论是应保存 N（N 为采样点个数）个运算参数 W_N^r，但由于每一次蝶形运算结果（如 $x_1(k)+W_N^kX_2(k)$，$x_1(k)-W_N^kX_2(k)$）具有对称性，因此 RAM3 只需要保存 N/2 个W_N^r即可。

2、1、1 奇数轮蝶形运算

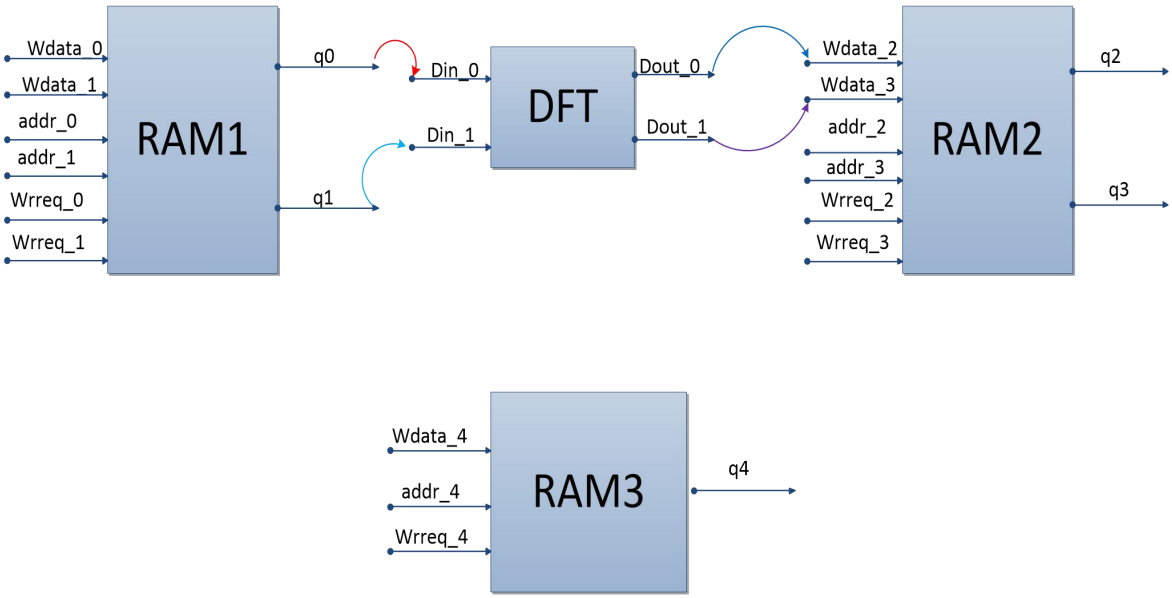


图 3 第奇数轮蝶形运算流图

如图 3 所示，RAM1 首先根据计数器给出的两个点的地址（如地址 0，地址 1）进行数据读操作，然后将数据（如$x_1(k)$和$x_2(k)$）送进 DFT 模块进行运算，最后 RAM2 将 DFT 模块输出的数据（如$x_1(k)+W_N^k X_2(k)$，$x_1(k)-W_N^k X_2(k)$）按照原来的地址顺序进行写操作，直到 RAM1 全部读完 N 个数据，并且 RAM2 全部写完 N 个数据后，则第一轮蝶形运算计算完毕。

2、1、2 偶数轮蝶形运算

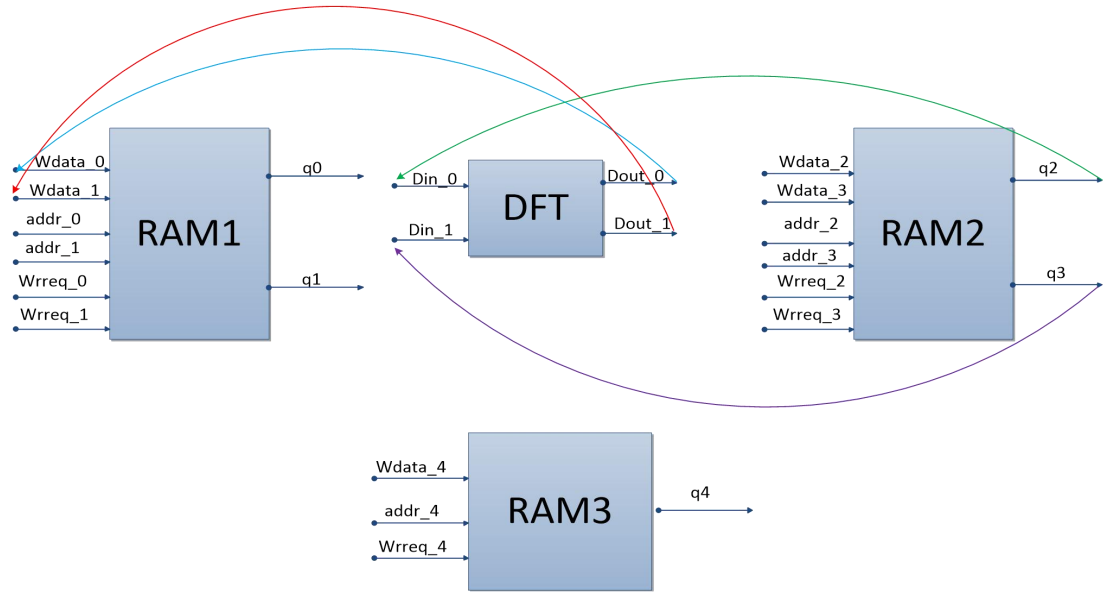


图 4　第偶数轮蝶形运算流图

偶数轮运算跟奇数轮运算相似，唯一的不同就是：读取 RAM 由 RAM1 改为 RAM2，写 RAM 由 RAM2 改为 RAM1。

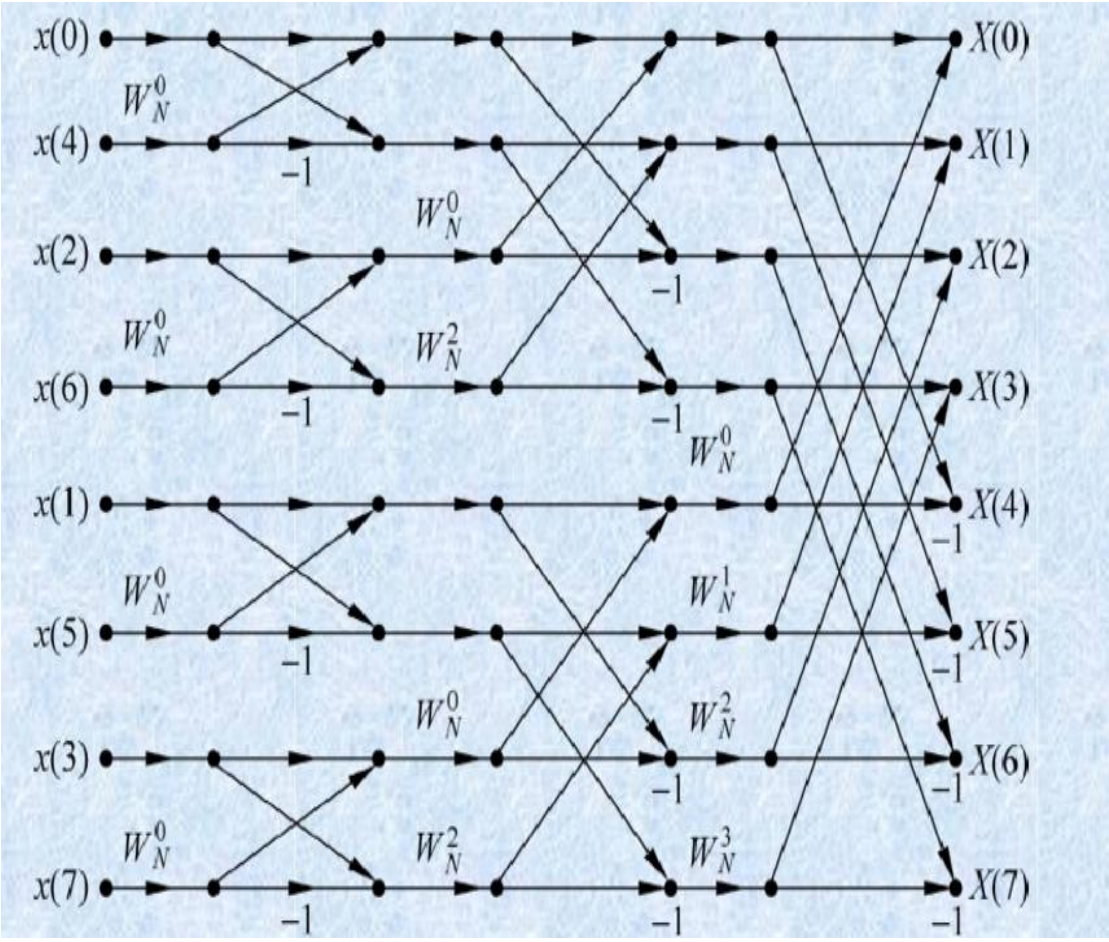
RAM1 与 RAM2 按照这样的读写交替顺序，直至历遍完 n 轮蝶形运算（n 为蝶形运算一共要计算的轮数）。

2、2　计数器架构设计

由于需要依次读取和写入 RAM1 和 RAM2，并且还要经过 N 轮的运算，很明显需要运用到计数器。

计数器架构，关乎到整个设计的可靠性和至简性，因此是重中之中的设计。按照至简设计法的建议，需要用到 N 轮运算，这需要一个计数器但每轮的计数器如何设计呢？

由于这些计数器主要是用于产生读写地址的，所以我们需要仔细分析地址的规律。我们以 8 点的 FFT 为例进行分析。



观察上图，每一轮取址如表 1 所示:

蝶形运算第几轮	运算节点	第一次蝶形运算	第二次蝶形运算	第三次蝶形运算	第四次蝶形运算
1	X_1(k)的地址	0	2	4	6
	X_2(k)的地址	1	3	5	7
2	X_1(k)的地址	0	1	4	5
	X_2(k)的地址	2	3	6	7
3	X_1(k)的地址	0	1	2	3
	X_2(k)的地址	4	5	6	7

表 1　N 为 8 的蝶形运算每一轮取址

蝶形运算每一轮每一次的取地址满足什么关系呢，如何才能在 FPGA 设计中实现如表 1 的取地址运算，观察上表，我们可以发现如下规律:

第几级蝶形运算	X_1(k)的地址			
	第一次	第二次	第三次	第四次
第一级	$0=0+0*2^1$	$2=0+1*2^1$	$4=0+2*2^1$	$6=0+3*2^1$
第二级	$0=0+0*2^2$	$1=1+0*2^2$	$4=0+1*2^2$	$5=1+1*2^2$
第三级	$0=0+0*2^3$	$1=1+0*2^3$	$2=2+0*2^3$	$3=3+0*2^3$

表 2　X_1(k)的取址

第几级蝶形运算	X_2(k)的地址			
	第一次	第二次	第三次	第四次
第一级	$1=2^0+0+0*2^1$	$3=2^0+0+1*2^1$	$5=2^0+0+2*2^1$	$7=2^0+0+3*2^1$
第二级	$2=2^1+0+0*2^2$	$3=2^1+1+0*2^2$	$6=2^1+0+1*2^2$	$7=2^1+1+1*2^2$
第三级	$4=2^2+0+0*2^3$	$5=2^2+1+0*2^3$	$6=2^2+2+0*2^3$	$7=2^2+3+0*2^3$

表 3　　X_2(k)的取址

根据表 2、表 3，可得到$X_1(k)_{地址}$与$X_2(k)_{地址}$与数组[a],[b],[c]有关的表达式

$$X_1(k)_{地址} = [a] + [b] * 2^{[c]};$$

$$X_2(k)_{地址} = X_1(k)_{地址} + 2^{[c]-1}; \quad （式 1）$$

通过上面的观察，按照明德扬的计数器架构建议，可设计三个计数器 cnt0,cnt1,cnt2 分别表示数组[a],[b],[c]，因此可将式 1 变为：

$$X_1(k)_{地址} = cnt0 + cnt1 * 2^{cnt2+1};$$

$$X_2(k)_{地址} = X_1(k)_{地址} + 2^{cnt2}; \quad （式 2）$$

各个计数器每一轮的结束条件为：

$$cnt0 = 2^{cnt2} - 1;$$

$$cnt1 = 2^{n-1-cnt2} - 1;$$

$$cnt2 = n - 1; \quad （式 3）$$

其中 n 为蝶形运算一共要计算的轮数，如采样点数 N 为 8 时，则一共要进行三轮运算。通过这三个**简易的计数器**设计，就能实现复杂的 DIT-FFT 蝶形运算**取址操作**。

终上所述，无论是模块划分、计数器设计、还是乒乓操作的读写处理，都始终基于“至简设计”的原则，用简易的代码结构就能实现复杂的 DIT-FFT 蝶形运算，代码设计风格极其简洁，详细可参考附录代码。

3、至简设计代码实现（附录部分代码）

```
1   //四段式状态机
2
3   //第一段：同步时序 always 模块，格式化描述次态寄存器迁移到现态寄存器
4   always@(posedge clk or negedge rst_n)begin
5       if(!rst_n)begin
6           state_c <= IDLE;
7       end
8       else begin
9           state_c <= state_n;
10      end
11  end
12
13  //第二段：组合逻辑 always 模块，描述状态转移条件判断
14  always@(*)begin
15      case(state_c)
16          NOP:begin
17              if(nop2precharge_start)begin
```

```
                    state_n = PRECHARGE;
                end
                else begin
                    state_n = state_c;
                end
            end
            PRECHARGE:begin
                if(precharge2autorefresh_start)begin
                    state_n = AUTOREFRESH;
                end
                else if(precharge2idle_start)begin
                    state_n = IDLE;
                end

                else begin
                    state_n = state_c;
                end
            end
            AUTOREFRESH:begin
                if(autorefresh2autorefresh_start)begin
                    state_n = AUTOREFRESH;
                end
                else if(autorefresh2loadmode_start)begin
                    state_n = LOADMODE;
                end
                else if(autorefresh2idle_start)begin
                    state_n = IDLE;
                end
            LOADMODE:begin
                if(loadmode2idle_start)begin
                    state_n = IDLE;
                end
                else begin
                    state_n = state_c;
                end
            end
            IDLE:begin
                if(idle2active_start)begin
                    state_n = ACTIVE;
                end
                else if(idle2autorefresh_start)begin
                    state_n = AUTOREFRESH;
                end

                else begin
                    state_n = state_c;
                end
            end
            ACTIVE:begin
                if(active2read_start)begin
                    state_n = READ;
                end
                else if(active2write_start)begin
                    state_n = WRITE;
                end

                else begin
```

```
                    state_n = state_c;
                end
            end
            READ:begin
                if(read2precharge_start)begin
                    state_n = PRECHARGE;
                end
                else begin
                    state_n = state_c;
                end
            end
            WRITE:begin
                if(write2precharge_start)begin
                    state_n = PRECHARGE;
                end
                else begin
                    state_n = state_c;
                end
            end
            default:begin
                state_n = IDLE;
            end
        endcase
    end
    //第三段：设计转移条件
    assign nop2precharge_start              = state_c==NOP              && end_cnt;

    assign precharge2autorefresh_start      = state_c==PRECHARGE        && init_flag==1
    &&end_cnt;
    assign precharge2idle_start             = state_c==PRECHARGE        && init_flag==0
    &&end_cnt;

    assign autorefresh2autorefresh_start =  state_c==AUTOREFRESH        && init_flag==1
    && init_auto_flag==1&&end_cnt;
    assign autorefresh2loadmode_start       = state_c==AUTOREFRESH      && init_flag==1
    && init_auto_flag==0&&end_cnt;

    assign autorefresh2idle_start           = state_c==AUTOREFRESH      && init_flag==0
    && end_cnt;

    assign loadmode2idle_start              = state_c==LOADMODE         && end_cnt;

    assign idle2active_start                = state_c==IDLE             && ref_req ==
    0&&(wr_req||rd_req);
    assign idle2autorefresh_start           = state_c==IDLE             && ref_req ==1 ;

    assign active2read_start                = state_c==ACTIVE           && rd_flag==1
    &&end_cnt;
    assign active2write_start               = state_c==ACTIVE           && rd_flag==0
    &&end_cnt;

    assign read2precharge_start             = state_c==READ             && end_cnt;
    assign write2precharge_start            = stae_c == WRITE           && end_cnt;

    always  @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
            init_flag <= 1;
```

```
    end
    else if(loadmode2idle_start)begin
        init_flag <= 0;
    end
end

always @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        init_auto_flag <= 1;
    end
    else if(autorefresh2autorefresh_start)begin
        init_auto_flag <= 0;
    end
end

//刷新请求
always @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        ref_req <= 0;
    end
    else if(end_cnt_self)begin
        ref_req <= 1;
    end
    else if(idle2autorefresh_start)begin
        ref_req <= 0;
    end
end

//写响应
assign wr_ack = idle2active_start==1&&
(wr_req==1&&((rd_flag==1)||(rd_flag==0&&rd_req==0)));
//读响应
assign rd_ack = idle2active_start==1&&
(rd_req==1&&((rd_flag==0)||(rd_flag==1&&wr_req==0)));

always @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        rd_flag <= 0;
    end
    else if(rd_flag==0&&rd_ack==1)begin
        rd_flag <= 1;
    end
    else if(rd_flag==1&&wr_ack==1)begin
        rd_flag <= 0;
    end
end



always @(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        cnt_self <= 0;
    end
    else if(add_cnt_self)begin
        if(end_cnt_self)
            cnt_self <= 0;
        else
```

```
189                 cnt_self <= cnt_self + 1;
190         end
191 end
192
193 assign add_cnt_self = init_flag==0;
194 assign end_cnt_self = add_cnt_self && cnt_self==1562-1 ;
195
196
197 always @(posedge clk or negedge rst_n)begin
198     if(!rst_n)begin
199         cnt <= 0;
200     end
201     else if(add_cnt)begin
202         if(end_cnt)
203             cnt <= 0;
204         else
205             cnt <= cnt + 1;
206     end
207 end
208
209 assign add_cnt = state_c!=IDLE;
210 assign end_cnt = add_cnt && cnt==x-1 ;
211
212  always   @(*)begin
213         if(state_c== NOP)begin
214             x =   INITIATE;
215         end
216         else if(state_c== PRECHARGE)begin
217             x =   TRP;
218         end
219         else if(state_c == AUTOREFRESH)begin
220             x = TRC;
221         end
222         else if(state_c == LOADMODE)begin
223             x = TMRD;
224         end
225         else if(state_c ==ACTIVE)begin
226             x = TRCD ;
227         end
228         else begin
229             x = 256 ;
230         end
231 __
232     end
233
234   always   @(posedge clk or negedge rst_n)begin
235       if(rst_n==1'b0)begin
236           cke <= 1;
237       end
238       else begin
239           cke <= 1;
240       end
241   end
242
243  assign command = {cs,ras,cas,we};
244
245   assign nop             = 4'b1000;
```

```
246   assign precharge       = 4'b0010;
247   assign autorefresh     = 4'b0001;
248   assign loadmode        = 4'b0000;
249   assign active          = 4'b0011;
250   assign read            = 4'b0101;
251   assign write           = 4'b0100;
252
253   always  @(posedge clk or negedge rst_n)begin
254       if(rst_n==1'b0)begin
255           command <= 0;
256       end
257       else
258 if(nop2precharge_start||write2precharge_start||read2precharge_start)begin
259           command <= precharge;
260       end
261       else
262 if(precharge2autorefresh_start||autorefresh2autorefresh_start||idle2autorefresh_sta
263 rt)begin
264           command <= autorefresh;
265       end
266       else if(autorefresh2loadmode_start)begin
267           command <= loadmode;
268       end
269       else if(idle2active_start)begin
270           command <= active;
271       end
272       else if(active2read_start)begin
          command <= read;
      end
      else if(active2write_start)begin
          command <= write;
      end
      else begin
          command <= nop;
      end

  end

  always  @(posedge clk or negedge rst_n)begin
      if(rst_n==1'b0)begin
          dqm <= 0;
      end
      else if(init_flag==1)begin
          dqm <= 2'b11;
      end
      else begin
          dqm <= 0;
      end
  end

always  @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        dq_out <= 0;
    end
    else begin
        dq_out <= wdata;
    end
```

```
  end

  always   @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
              dq_out_en <= 0;
        end
        else if(Active2Write_start)begin
              dq_out_en <= 1;
        end
        else if(write2precharge_start)begin
              dq_out_en <= 0;
        end
  end


___
   always   @(posedge clk or negedge rst_n)begin                    //256 拍
         if(rst_n==1'b0)begin
               rdata_flag <= 0;
         end
         else if(active2read_start)begin
               rdata_flag <= 1;
         end
         else if(read2precharge_start)begin
               rdata_flag <= 0;
         end
   end

   always   @(posedge clk or negedge rst_n)begin
         if(rst_n==1'b0)begin
               rdata_flag_ff0 <= 0;
               rdata_flag_ff1 <= 0;
               rdata_flag_ff2 <= 0;
         end
         else begin
               rdata_flag_ff0 <= rdata_flag;
               rdata_flag_ff1 <= rdata_flag_ff0;
               rdata_flag_ff2 <= rdata_flag_ff1;

_______
         end
   end


   always   @(posedge clk or negedge rst_n)begin
         if(rst_n==1'b0)begin
               rdata <= 0;
         end
         else begin
               rdata <= dq_in;
         end

   end

   always   @(posedge clk or negedge rst_n)begin
         if(rst_n==1'b0)begin
               rdata_vld <= 0;
         end
         else begin
```

```
            rdata_vld <= rd_flag_ff2;
        end
    end

    always    @(posedge clk or negedge rst_n)begin      //锁存地址 waddr;
        if(rst_n==1'b0)begin
            waddr_ff0 <= 0;
        end
        else if(idle2active_start)begin
            waddr_ff0 <= waddr;
        end
    end

    always    @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
            addr <= 0;
        end
        else if(nop2precharge_startt)begin
            addr <=12'b0100_0000_0000;
        end
        else if(autorefresh2loadmode_start)begin
            addr <= 12'b00_1_00_010_0_111;//latency Mode 为 010 的情况
        end
        else if(idle2active_start)begin
            addr[7:0] <=    waddr[19:8];      //行地址
        end
        else if(active2write_start||active2read_start)begin
            addr[7:0] <=    waddr_ff0[7:0];       //列地址
        end

    always    @(posedge clk or negedge rst_n)begin
        if(rst_n==1'b0)begin
            bank <=0;
        end
        else if(nop2precharge_start)begin
            bank <= 2'b11;
        end
        else if(idle2active_start)begin
            bank <= waddr[21:20];
        end
        else if(active2read_start||active2write_start)begin
            bank < waddr_ff0[21:20];
        end
        else begin
            bank <= 0;
        end
    end
```