ct_lsu_pfu_gsdb.v

概要

- 模块内容：全局预取模式的步幅计算和检查的控制逻辑
- 步幅判断最核心的部分为Generate state machine，它是一个有四个状态的有限状态机。首先可以看四个状态的含义，然后看一下四个状态的编码方式，再分析敏感事件表中的信号，明确满足哪个信号时状态会发生改变，最后看一下如何回到初始态。
- 在GET_STRIDE和CHECK_STRIDE状态，主要调用ct_lsu_pfu_sdb_cmp模块进行计算。
- 在MONITOR_STRIDE状态时，主要由confidence部分决定。分析何时置信度会加减和重置。
- Maintain newest iid
- gated_clk_cell用于在不同的操作阶段启用或禁用时钟，降低功耗提升性能
- 输出信号一共六个，含义分别如下：
  entry_addr_cmp_info_vld　　//完成地址信息比较，清空三个entry
  pfu_gsdb_gpfb_create_vld;　//check阶段成功
  pfu_gsdb_gpfb_pop_req;　　//monitor阶段失败
  [10:0] pfu_gsdb_stride;　　//步幅[10:0]
  pfu_gsdb_stride_neg;　　//步幅[11]
  [6 :0] pfu_gsdb_strideh_6to0;　　//步幅[6:0]
- 关于lsu中的一些问题的解释

Generate state machine

```
parameter IDLE              = 4'b0000,
          GET_STRIDE        = 4'b1001,
          CHECK_STRIDE      = 4'b1010,
          MONITOR_STRIDE    = 4'b1100;

always @(posedge pfu_gsdb_clk or negedge cpurst_b)
begin
    if (!cpurst_b)           //复位信号
    pfu_gsdb_state[3:0] <=  IDLE;
  else if(pfu_pop_all_vld) //某些条件下（如流水线刷新）控制预取请求
    pfu_gsdb_state[3:0] <=  IDLE;
  else
    pfu_gsdb_state[3:0] <=  pfu_gsdb_next_state[3:0];
end
assign pfu_gsdb_vld = pfu_gsdb_state[3];   //表示非初始态

//IDLE,GET,CHECK,MONITOR分别编号为0123，下面敏感列表括号内表示该信号在哪个状态改变时会产生影响
//如果均产生影响，则用*表示
always @( confidence_min               //置信度为0          (3)
       or pfu_gsdb_normal_stride       //地址差不为0不大于2k 且 地址差相等cmp(1)
       or pfu_gsdb_create_vld          //预取的创建         (0)
       or pfu_gsdb_addr_cmp_info_vld   //完成地址信息的比较  (12)
       or pfu_gpfb_vld                 //gpfb(3)
       or pfu_gsdb_state[3:0]          //产生步幅有限状态机的当前状态(*)
       or pfu_gsdb_check_stride_success) //步幅检查成功信号  (23)
begin
pfu_gsdb_next_state[3:0]  = IDLE;
case(pfu_gsdb_state[3:0])
  IDLE:
    if(pfu_gsdb_create_vld)
      pfu_gsdb_next_state[3:0]  = GET_STRIDE;
    else
      pfu_gsdb_next_state[3:0]  = IDLE;
  GET_STRIDE:
```

```
    if(pfu_gsdb_addr_cmp_info_vld &&  pfu_gsdb_normal_stride)
      pfu_gsdb_next_state[3:0]  = CHECK_STRIDE;
    else
      pfu_gsdb_next_state[3:0]  = GET_STRIDE;
  CHECK_STRIDE:
    if(pfu_gsdb_addr_cmp_info_vld &&  pfu_gsdb_check_stride_success)
      pfu_gsdb_next_state[3:0]  = MONITOR_STRIDE;
    else if(pfu_gsdb_addr_cmp_info_vld &&  !pfu_gsdb_check_stride_success)
      pfu_gsdb_next_state[3:0]  = GET_STRIDE;
    else
      pfu_gsdb_next_state[3:0]  = CHECK_STRIDE;
  MONITOR_STRIDE:
    if(pfu_gsdb_addr_cmp_info_vld
         &&  (!pfu_gsdb_check_stride_success
              && confidence_min
             ||  !pfu_gpfb_vld))
      pfu_gsdb_next_state[3:0]  = GET_STRIDE;
    else
      pfu_gsdb_next_state[3:0]  = MONITOR_STRIDE;
  default:
    pfu_gsdb_next_state[3:0]  = IDLE;
endcase
// &CombEnd; @165
end
assign pfu_gsdb_state_is_get_stride     = pfu_gsdb_state[0];
assign pfu_gsdb_state_is_check_stride   = pfu_gsdb_state[1];
assign pfu_gsdb_state_is_monitor_stride = pfu_gsdb_state[2];
```

以上是一个产生步幅的有限状态机，共有4个状态：

- **IDLE**：该状态为初始状态，在该状态下等待预取的开启，当机器模式隐式操作寄存器（MHINT）开启与预取相关的位时，则会跳转到 **GET_STRIDE** 进行步幅的计算。
- **GET_STRIDE**：等待步幅的计算，当步幅计算完成后（根据 `entry_addr_cmp_info_vld` 和 `normal_stride` 信号判断）跳转到**CHECK_STRIDE** 进行步幅的检查。
- **CHECK_STRIDE**：检查步幅，当步幅检查成功后（根据 `check_stride_success` 信号判断）跳转到**MONITOR_STRIDE** 监视步幅，同时会将置信度重置为2'b10，同时会向gpfb（主要用于控制向总线发出预取请求）**发出请求**，并将步幅传递；当步幅检查发现不一致时，跳转到 **GET_STRIDE** 重新计算步幅。
- **MONITOR_STRIDE**：监视步幅，在该状态下也会不断地检查步幅，每成功检查一次，置信度会加一；每一次检查不成功时，置信度会减一，当置信度减为0时，会跳转回 **GET_STRIDE** 重新等待下一次步幅计算，同时需要向gpfb发出pop请求，及时停止当前预取。

根据状态的编码可得，用四位二进制进行编码。由最高位判断是否在初始态，若最高位为1，说明正在产生或监视步幅。其余三位采用独热码的方式进行区别状态。

ct_lsu_pfu_sdb_cmp

```
ct_lsu_pfu_sdb_cmp  x_ct_lsu_pfu_gsdb_cmp (
//input
  .cp0_lsu_icg_en               (cp0_lsu_icg_en              ),
  .cp0_yy_clk_en                (cp0_yy_clk_en               ),
  .cpurst_b                     (cpurst_b                    ),
  .entry_addr0_act              (pfu_gsdb_addr0_act          ),  //load指令比newest_pf_inst存储的新
  .entry_clk                    (pfu_gsdb_clk                ),
  .entry_create_dp_vld          (pfu_gsdb_create_dp_vld      ),
  .entry_create_gateclk_en      (pfu_gsdb_create_gateclk_en  ),
  .entry_pf_inst_vld            (pfu_gsdb_pf_inst_vld        ),  //load指令预取有效
  .entry_stride_keep            (monitor_with_confidence     ),  //监视置信度
  .entry_vld                    (pfu_gsdb_vld                ),  //处于非IDLE状态
  .forever_cpuclk               (forever_cpuclk              ),
  .ld_da_iid                    (ld_da_iid                   ),
```

```
  .pad_yy_icg_scan_en            (pad_yy_icg_scan_en          ),
  .pipe_va                       (ld_da_pfu_va                ),
  .rtu_yy_xx_commit0             (rtu_yy_xx_commit0           ),
  .rtu_yy_xx_commit0_iid         (rtu_yy_xx_commit0_iid       ),
  .rtu_yy_xx_commit1             (rtu_yy_xx_commit1           ),
  .rtu_yy_xx_commit1_iid         (rtu_yy_xx_commit1_iid       ),
  .rtu_yy_xx_commit2             (rtu_yy_xx_commit2           ),
  .rtu_yy_xx_commit2_iid         (rtu_yy_xx_commit2_iid       ),
  .rtu_yy_xx_flush               (rtu_yy_xx_flush             ),
//output
  .entry_addr_cmp_info_vld       (pfu_gsdb_addr_cmp_info_vld   ),  //完成地址信息比较，在123阶段都会用到
  .entry_check_stride_success    (pfu_gsdb_check_stride_success),  //检查步幅成功，在23阶段用到
  .entry_normal_stride           (pfu_gsdb_normal_stride       ),  //步幅满足要求且相同，在1阶段用到
  .entry_stride                  (pfu_gsdb_stride              ),  //模块output 步幅[10:0] 表示范围2k
  .entry_stride_neg              (pfu_gsdb_stride_neg          ),  //模块output 步幅[11]，用于表示步幅正负，1表示
  .entry_strideh_6to0            (pfu_gsdb_strideh_6to0        )   //模块output 步幅[6:0] 表示范围128byte，缓存
);

assign entry_stride_neg_ge_cache_line = !(&entry_stride[10:6]);

assign entry_stride_pos_ge_cache_line = |entry_stride[10:6];

assign entry_stride_ge_cache_line     = entry_stride_neg
                                        ? entry_stride_neg_ge_cache_line
                                        : entry_stride_pos_ge_cache_line;

assign entry_strideh_6to0[6:0]        = entry_stride_ge_cache_line
                                        ? entry_stride[6:0]
                                        : 7'h40;
//这里需要判断是否跨过了一个缓存行，正步幅时，通过将高位每一位相或即可判断。如果跨过一个缓存行，则按照原来的步幅，若未跨过，置为
```

这里只介绍GSDB 特有的状态和配置信号，内部具体信号含义可参照王一鸣ct_lsu_pfu_sdb

confidence（MONITOR_STRIDE）

```
always @(posedge pfu_gsdb_clk or negedge cpurst_b)
begin
  if (!cpurst_b)
    pfu_gsdb_pop_confidence[1:0] <=  2'b0;
  else if(confidence_reset)
    pfu_gsdb_pop_confidence[1:0] <=  2'b10;
  else if(confidence_sub_vld)
    pfu_gsdb_pop_confidence[1:0] <=  pfu_gsdb_pop_confidence[1:0] - 2'b01;
  else if(confidence_add_vld)
    pfu_gsdb_pop_confidence[1:0] <=  pfu_gsdb_pop_confidence[1:0] + 2'b01;
end

assign confidence_max = (pfu_gsdb_pop_confidence[1:0] == 2'b11);
assign confidence_min = (pfu_gsdb_pop_confidence[1:0] == 2'b00);

assign confidence_reset   = pfu_gsdb_state_is_check_stride //处于check状态
                            && pfu_gsdb_addr_cmp_info_vld  //完成地址信息的比较
                            && pfu_gsdb_check_stride_success; //成功检查步幅

assign confidence_sub_vld = pfu_gsdb_state_is_monitor_stride  //处于monitor状态
                            && pfu_gsdb_addr_cmp_info_vld     //完成地址信息的比较
                            && !pfu_gsdb_check_stride_success //检查步幅失败
                            && !confidence_min;               //置信度未降至0
```

```
assign confidence_add_vld = pfu_gsdb_state_is_monitor_stride
                            && pfu_gsdb_addr_cmp_info_vld
                            && pfu_gsdb_check_stride_success  //检查步幅成功
                            && !confidence_max;               //置信度未达到最大

assign monitor_with_confidence = pfu_gsdb_state_is_monitor_stride //处于monitor状态
                                 && pfu_gpfb_vld //gpfb
                                 && !confidence_min;          //置信度未降至0
```

当处于检查步幅的状态时，将置信度重置为2'b10。在监视步幅的状态下，每成功检查步幅一次（`pfu_gsdb_check_stride_success`为1），则将置信度加一；若检查发现步幅与之前的步幅不一致，则置信度减一。

Maintain newest iid

```
always @(posedge pfu_gsdb_clk or negedge cpurst_b)
begin
  if (!cpurst_b)
    pfu_gsdb_newest_pf_inst_vld <=  1'b0;
  else if(pfu_gsdb_create_dp_vld || pfu_gsdb_newest_pf_inst_flush_uncmit) //IDLE状态 或 刷新未提交
    pfu_gsdb_newest_pf_inst_vld <=  1'b0;
  else if(pfu_gsdb_vld  &&  pfu_gsdb_pf_inst_vld)     //非IDLE状态，预取指令有效
    pfu_gsdb_newest_pf_inst_vld <=  1'b1;
end

//pfu_gsdb_newest_pf_inst_set有效时，用加载指令id更新最新的预取指令id
always @(posedge pfu_gsdb_pf_inst_vld_clk or negedge cpurst_b)
begin
  if (!cpurst_b)
    pfu_gsdb_newest_pf_inst_iid[6:0]  <=  7'b0;
  else if(pfu_gsdb_newest_pf_inst_set)
    pfu_gsdb_newest_pf_inst_iid[6:0]  <=  ld_da_iid[6:0];
end

//满足最新的预取指令提交的条件，最新的预取指令提交标志设为1
always @(posedge pfu_gsdb_clk or negedge cpurst_b)
begin
  if (!cpurst_b)
    pfu_gsdb_newest_pf_inst_cmit  <=  1'b0;
  else if(pfu_gsdb_newest_pf_inst_set)
    pfu_gsdb_newest_pf_inst_cmit  <=  1'b0;
  else if(pfu_gsdb_newest_pf_inst_cmit_set)
    pfu_gsdb_newest_pf_inst_cmit  <=  1'b1;
end

//============================================================
//              Maintain newest iid
//============================================================
//-------------------------older----------------------------
//比较newest_pf_inst存放的最新iid和load指令的iid
ct_rtu_compare_iid  x_lsu_gsdb_newest_inst_cmp (
  .x_iid0                                     (pfu_gsdb_newest_pf_inst_iid[6:0]            ),
  .x_iid0_older                               (pfu_gsdb_newest_pf_inst_iid_older_than_ld_da),  //比较结果
  .x_iid1                                     (ld_da_iid[6:0]                              )
);

//提交命中标志
assign pfu_gsdb_newest_pf_inst_cmit_hit0  = {rtu_yy_xx_commit0,rtu_yy_xx_commit0_iid[6:0]}
                                            ==  {1'b1,pfu_gsdb_newest_pf_inst_iid[6:0]};
assign pfu_gsdb_newest_pf_inst_cmit_hit1  = {rtu_yy_xx_commit1,rtu_yy_xx_commit1_iid[6:0]}
```

```
                                          ==  {1'b1,pfu_gsdb_newest_pf_inst_iid[6:0]};
assign pfu_gsdb_newest_pf_inst_cmit_hit2  = {rtu_yy_xx_commit2,rtu_yy_xx_commit2_iid[6:0]}
                                          ==  {1'b1,pfu_gsdb_newest_pf_inst_iid[6:0]};
//提交的条件
assign pfu_gsdb_newest_pf_inst_cmit_set   = (pfu_gsdb_newest_pf_inst_cmit_hit0
                                          ||  pfu_gsdb_newest_pf_inst_cmit_hit1
                                          ||  pfu_gsdb_newest_pf_inst_cmit_hit2)    //三者有一个命中
                                          &&  pfu_gsdb_newest_pf_inst_vld;
//最新预取指令是否早于加载指令
assign pfu_gsdb_newest_pf_inst_older_than_ld_da = pfu_gsdb_newest_pf_inst_vld
                                            &&  (pfu_gsdb_newest_pf_inst_iid_older_than_ld_da
                                                ||  pfu_gsdb_newest_pf_inst_cmit);

//-------------------newest_pf_inst_set---------------------
//更新最新预取指令ID的条件，newest_pf_inst存储所有经过da阶段的指令中最新指令的iid
assign pfu_gsdb_newest_pf_inst_set  = pfu_gsdb_vld                          //非IDLE状态
                                    &&  pfu_gsdb_pf_inst_vld                //预取指令有效
                                    &&  (!pfu_gsdb_newest_pf_inst_vld       //newest_pf_inst存储非最新指令
                                        ||  pfu_gsdb_newest_pf_inst_older_than_ld_da);
//未提交的指令遇到刷新
assign pfu_gsdb_newest_pf_inst_flush_uncmit = rtu_yy_xx_flush               //刷新操作
                                        &&  !pfu_gsdb_newest_pf_inst_cmit;  //指令未提交
```

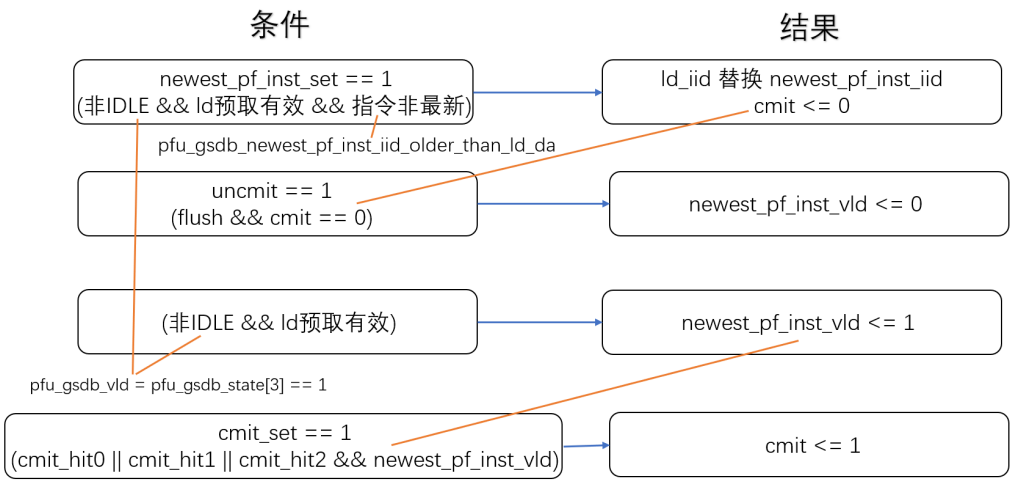
valid信号总结

```
//预取的创建，脱离IDLE状态
assign pfu_gsdb_create_vld          = !pfu_gsdb_vld        //处于IDLE状态
                                    &&  cp0_yy_dcache_pref_en;  //外部使能信号
//非IDLE状态
assign pfu_gsdb_vld = pfu_gsdb_state[3];                    //处于非IDLE状态
//预取有效
assign pfu_gsdb_pf_inst_vld   = pfu_gsdb_vld                //处于非IDLE状态
                              &&  ld_da_pfu_pf_inst_vld;  //模块输入信号，load指令预取有效

//这个信号和pfu_gsdb_pf_inst_vld为1的条件是一样的，只是需要时钟边沿触发，感觉是为了可以写多个条件改变它的值
if(pfu_gsdb_vld  &&  pfu_gsdb_pf_inst_vld)
    pfu_gsdb_newest_pf_inst_vld <=  1'b1;
```

条件信号set总结

```
//指令提交的条件
assign pfu_gsdb_newest_pf_inst_cmit_set   = (pfu_gsdb_newest_pf_inst_cmit_hit0
                                          ||  pfu_gsdb_newest_pf_inst_cmit_hit1
                                          ||  pfu_gsdb_newest_pf_inst_cmit_hit2)    //三者有一个命中
                                          &&  pfu_gsdb_newest_pf_inst_vld;              //newest_pf_inst存
//满足此条件，pfu_gsdb_newest_pf_inst_cmit  <=  1'b1;

//更新最新预取指令ID的条件，newest_pf_inst存储所有经过da阶段的指令中最新指令的iid
assign pfu_gsdb_newest_pf_inst_set  = pfu_gsdb_vld                          //非IDLE状态
                                    &&  pfu_gsdb_pf_inst_vld                //预取指令有效
                                    &&  (!pfu_gsdb_newest_pf_inst_vld       //newest_pf_inst存储非最新指令
                                        ||  pfu_gsdb_newest_pf_inst_older_than_ld_da);
//满足此条件，  pfu_gsdb_newest_pf_inst_iid[6:0]  <=  ld_da_iid[6:0];
```

信号关联图：蓝色箭头具有因果关系，橙色直线代表数据依赖（由于我无法真正理解commit，看了很久很久也不懂，这部分就卡住了）



该部分应该是实现了以下部分：

1. 通过ct_rtu_compare_iid子模块比较newest_pf_inst的iid和load指令的iid，使得newest_pf_inst维持最新的load指令
2. 将newest_pf_inst_vld信号作为指令提交的条件之一，如果遇到流水线刷新flush信号，将newest_pf_inst_vld信号置0，不提交指令

gated_clk_cell

```
assign pfu_gsdb_clk_en  = pfu_gsdb_vld
                        ||  pfu_gsdb_create_gateclk_en;

gated_clk_cell  x_lsu_pfu_gsdb_gated_clk (
  .clk_in             (forever_cpuclk    ),
  .clk_out            (pfu_gsdb_clk      ),
  .external_en        (1'b0              ),
  .global_en          (cp0_yy_clk_en     ),
  .local_en           (pfu_gsdb_clk_en   ),
  .module_en          (cp0_lsu_icg_en    ),
  .pad_yy_icg_scan_en (pad_yy_icg_scan_en)
);

assign pfu_gsdb_pf_inst_vld_clk_en = pfu_gsdb_pf_inst_vld;

gated_clk_cell  x_lsu_pfu_gsdb_pf_inst_vld_gated_clk (
  .clk_in                     (forever_cpuclk             ),
  .clk_out                    (pfu_gsdb_pf_inst_vld_clk   ),
  .external_en                (1'b0                       ),
  .global_en                  (cp0_yy_clk_en              ),
  .local_en                   (pfu_gsdb_pf_inst_vld_clk_en),
  .module_en                  (cp0_lsu_icg_en             ),
  .pad_yy_icg_scan_en         (pad_yy_icg_scan_en         )
);
```

`gated_clk_cell` 模块通过在不同的操作阶段启用或禁用时钟，这种方法可以显著减少处理器的功耗；还可以减少不必要的计算和内存访问，从而提高整体性能。

Output

```
output          entry_addr_cmp_info_vld;     //完成地址信息比较，清空三个entry
output          pfu_gsdb_gpfb_create_vld;    //check阶段成功
output          pfu_gsdb_gpfb_pop_req;       //monitor阶段失败
output  [10:0]  pfu_gsdb_stride;             //步幅[10:0] 表示范围2k
```

```
output          pfu_gsdb_stride_neg;         //步幅[11]
output  [6 :0]  pfu_gsdb_strideh_6to0;       //步幅[6:0]

assign pfu_gsdb_gpfb_create_vld     = pfu_gsdb_state_is_check_stride       //处于检查状态
                                    &&  pfu_gsdb_addr_cmp_info_vld       //完成地址信息比较
                                    &&  pfu_gsdb_check_stride_success;   //检查步幅成功

assign pfu_gsdb_gpfb_pop_req        = pfu_gsdb_state_is_monitor_stride     //处于监视阶段
                                    &&  pfu_gsdb_addr_cmp_info_vld       //完成地址信息比较
                                    &&  confidence_min                   //置信度为0
                                    &&  !pfu_gsdb_check_stride_success;  //检查步幅失败
```

LSU

在指令译码单元中会将指令分解成微操作(`uop`)，访存流水线分为AG(Address generation)、DC(Data cache)、DA(Data align)、WB(Write back)四个阶段

- AG：完成地址计算 + 完成虚拟地址转换 + 判断指令是否已经提交
- DC：访问DCache判断是否命中 + 其它与流水线相关的检查（与我们关系不大）
- DA：从Dcache中的加载数据选择相应的字节并进行对齐
- WB：将加载数据写回到寄存器中

Q：为什么判断信号要在ld_da阶段？

A：指令在DC阶段时，如果缓存命中就会有hit信号，如果hit信号为低，说明为Cache miss。那么该指令需要在DA阶段写入Read buffer，并 在Read buffer中创建一个项，排队请求来自L2 Cache的数据

Q：为什么两个步幅加起来不能跨4k？

A：玄铁完成并行执行的方式是对基地址高位`vpn`进行虚拟地址转换获得`ppn`，所以，最终得到的物理地址为`ppn+基址低12位+偏移量`，但是如果`基址低12位+偏移量`大小超过了12位，或者变成了负值(即发生了正溢出或负溢出)，索引到的地址其实超出了这4KB，也就是超过了最终翻译到的这一页

Q：什么是指令的提交（commit）？

A：指令的提交与退休是指令在处理器中运行的重要状态，而这个状态也是由ROB进行管理的。**指令的提交（commit）**是指本条指令已经处于ROB头部的退休窗口（前三项）中，且位于这条指令前的所有指令都已经正常退休或正在正常退休。换言之，一条指令的提交意味着这条指令是安全的（一定会被执行的）。而**指令的退休（retire）**则是指令在执行后，且可以确定对处理器产生影响后，就从ROB中移除的过程。玄铁C910支持某些指令的退休先于其写回寄存器，这些指令通常是不会产生异常的指令，如算数指令。

Q：什么是ROB（Re-order Buffer）？
A：重排序缓存，见PPT中第8节