〔19〕中华人民共和国专利局

[11]公开号 CN 1051645A

〔12〕发明专利申请公开说明书

[21] 申请号 90107821.2

[51] Int.Cl[5]

H04J 3/02

〔43〕公开日 1991年5月22日

[22]申请日 90.9.15

[30]优先权

[32]89.9.15 [33]GB [31]8920980.3

[71]申请人 普列斯公司

地址 英国英格兰

[72]发明人 阿历克·约翰·特纳

[74]专利代理机构 中国专利代理有限公司

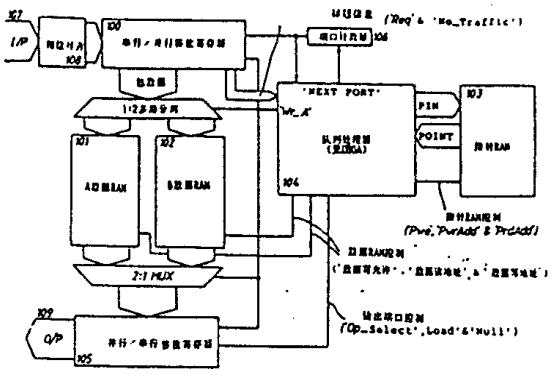
代理人 郭伟刚 匡少波

说明书页数: 24 附图页数: 10

[54]发明名称 异步时分多路开关系统

[57]摘要

本发明涉及ATDM开关系统，它包含用以按串行格式从若干输入端口接收输入的数据包(包括其路由信息)并以并行格式用这样方式送到随机存取存储器的串行到并行变换器，即，在存储器内由输入的包形成每个输出端口一个的队列。然后以并行格式依次从每个队列传送到并行到串行变换器，并以串行格式送到各相应输出端口。这样结构使得当存储器满了时该系统通过重写最长队列的数据而接纳输入的包，这种一种可在长时间运行时显著减少丢失包总数的技术。



权　利　要　求　书

1. 一种异步时分多路开关系统，它包含有安排用以接收包含串行形式的包含路由信息的输入的数据包并将之变换为并行形式的串行到并行变换器，将每个输入的数据包存储在所选址单元的存储器，用于将所述输入数据包排列成每个输出端口一个的队列的处理器，以及用以从所述数据队列接收并行形式的数据包并变换为串行形式送到各队列相应输出端口上的并行到串行变换器，并安排所述处理器保留每个所述队列中包个数的记录以便当存储器满了时，丢弃最长队列首部的包以留出输入的数据包的空间。

2. 如权利要求 1所述的系统，其特征在于，将串行到并行变换器划分为两个一半，将每个数据包划分为两个一半，并读到串行到并行变换器相应的一半上。

3. 如权利要求 2所述的系统，其特征在于，将存储器分为两个一半，安排每一半独立进行存取并用于存储来自串行到并行变换器相应一半的数据包的相应一半。

4. 如权利要求 3所述的系统，其特征在于，将并行到串行变换器划分为两个一半，安排每一半用于从存储器接收相应的一半的数据包以送到输出端口。

5. 如上述权利要求任一项所述的系统，其特征在于，输入数据为宽带数据块。

6. 如上述权利要求任一项所述的系统，其特征在于可用集成硅芯片加以实施。

7. 一种在异步时分多路开关系统中选择数据路由的方法，该方法包含这些步骤，将包括路由信息的输入的数据包从串行形式变换为并行形式，将输入数据的每个包送入所选址的存储器单元，将所述输入

的数据包排列成每个输出端口一个的队列，将所述数据包从并行形式变换为串行形式以送到和每个对列相应的输出端口，保留每个所述队列中数据包个数的记录，并在存储器满了时丢弃最长队列首部数据包以给输入的数据包留出空间。

8. 一种参考附图、基本如上文所述的异步时分多路开关系统。

9. 一种参考附图、基本如上文所述的在异步时分多路开关系统中选择数据路由的方法。

说　　明　　书

异步时分多路开关系统

本发明涉及异步时分多路(ATDM) 开关系统。

ATDM开关的功能是对通过多个输入端口进入开关的信息包选择通达由每个包中所含包标题信息说明的输出端口的路由，此外，提供对列设施以使该开关可应付多于一个输入端口上的包指定同一个输出端口的场合。实现该功能的简单方法是使用RAM，将RAM划分成分开的固定长度的队列，每个输出端口一个队列，一起使用的还有一些控制逻辑，该控制逻辑使用输入的包的标题域来确定该包应该存储在哪一队列。开关的输出侧简单地依次从每个队列尾端取走包并将它们送到相关输出端口。这种结构的问题在于它将每个队列限制为在 RAM内的固定量空间，这在不均匀传输条件下，由于有些队列已填满而另一些还有可用自由空间因此是低效率的。较好的方法是将队列构造成链接的 RAM 单元的链，该链可扩展到存储器中的空闲区。这里，同一队列中的各单元并不需要占用 RAM中的连续单元，因此系统须对每个包含一个包的单元存储该包队列的下一单元的地址，对每个空闲单元存储下一空闲单元的地址(这些空闲单元构成通常称为“自由表”的队列)，并对每个队列，存储该队列首、尾单元地址及该队列所含单元个数。例如这种布局，分配给给定队列的存储器于是就和指定给该队列相关输出端口的传输量成正比。但是，这种设计也存在问题，即，传输密度较高的通道可能将存储器装到某一点，使留给传输量较少的通道的队列空间很少或没有。一种解决方法是对每个队列保留最小量的存储器，但这样仍有原方案相同的缺点，即会有数据从较忙通道中丢失而 RAM 所保留的其它部分空间仍空闲着而理论上这些空间是可用的。所需要的是这样一种结构，如果并且在存储器满了时，给传输量较少的

通道存取该存储器的优先权。

根据本发明，提供一种异步时分多路开关系统，该系统包含：用以接收包含路由信息的串行形式的输入的数据包并将之变换为并行形式的串行到并行变换器，将每个输入数据包存储于其选址单元的存储器，用于将所述输入数据包排成每个输出端口一个的队列的处理器，以及，用以从所述数据队列接收并行形式的数据包并变换为串行形式以送到与每个队列相对应的输出端口上的并行到串行变换器，并且安排所述处理器保留每个所述队列中包的个数的记录，以便当存储器满了时，丢弃最长队列首部的包便为输入的数据包留出空间。

再根据本发明，提供一种在异步时分多路开关系统中选择数据路由的方法，该方法包含这些步骤，即，将包括路由信息的输入的数据包从串行变换为并行形式，将输入数据的每个包送入存储器的所选址单元，将所述输入的数据包排列成每个输出端口一个的队列，将所述数据包从并行形式变换为串行形式以送至和每个队列对应的输出端口，保留每个所述对列中数据包个数的记录，并在存储器满了时丢弃最长对列首部的数据包以便给输入的数据包留出空间。

最好将串行到并行变换器分裂为两半，将每个数据包分为两半并分别读入串行到并行变换器相应的一半，并将存储器分为两半，安排每一半可独立存取并用以存储来自相应一半的串行到并行变换器的相应一半的数据包。并行到串行变换器也分裂为两半，分别安排从存储器接收相应一半的数据包以送到输出端口。可用一个集成硅芯片实现实施本发明的系统。

下面，参考附图，通过实例对本发明作进一步说明，附图中：

图 1和图1A示出在包序列到达和离开期间实现四个先进／先出队列的“链接表”结构；

图2、2A和3示出说明为容纳输入的包而丢弃在最长队列首部的包

如何有效地减少了包丢失的ATDM开关系统的仿真结果；

图4示出ATDM开关系统；

图5示出在图4的系统中包到达和离开的时序图；以及

图6图示出实施本发明的、包括队列处理器的ATDM开关系统。

图1和1A示出在典型包序列到达和离开期间该“链接表”队列系统的结构变化。实现了标以A、B、C和D的四个独立的队列。矩形框表示以下存储器的物理区域：“数据”是8个字的RAM（随机存取存储器），其中每个字可保持一个信息包，“指针”是8个字的RAM，其中每个字具有指出“数据”RAM旁边地址的3位数值，“首”是保持每个队列（包括“自由表”）首部单元的RAM地址的5×3位寄存器列，“尾”是保持每个队列尾部单元的RAM地址的5×3位的寄存器列，“计数”是记录每个队列单元个数的5×4计数器列，而“最长队列”是指出当前4个数据队列中哪一个最长的2位寄存器。此外，该图的每一段也包括了在该时刻存在的队列的图形表示。符号“×”用于和RAM单元或寄存器中实际内容无关的地方。

图2、2A示出这里所述开关系统的PASCAL仿真所获得的结果。对用于处理包队列的两种算法进行了测试。在算法1中，当存储器满了时，将输入的包简单地丢弃，而在算法2中，通过牺牲处于最长队列首部的单元来接纳这些输入的包。通过说明到每个输出的相对传输量的传输密度比率，给出对列的传输分布。例如，如果将每16个包指定给输出端口0，端口1会有17、端口2有18个包等等，直至端口15共接收31个包。“总占有率”数值说明有效输入包的百分比，而将其余部分在其标题域作无效标志（即，空的）。从图2、2A可见，最忙队列（端口15）的损失大体上不受算法的选择的影响，而传输量较少的队列的损失急剧降低到数据总损失减少64%以上的程度。图3示出将这种改进应用于均匀分布的传输，并且这种分布在占有率水平的全部

范围内是一致的。尽管可期望较空队列丢弃较少数据(最后，只有较长且较忙队列被改写) ，但相反地较忙队列未受影响的原因并不能立即明白。其解释基于这样的事实，在任何给定周期内，当一个包到达时，如果没有另一包离开，则该存储器在当前所选输出端口的队列已空的情形下才会被装满。最可能变空的队列会是最短的队列，并且，由于确保这些队列不再丢失数据，可减少对列变空、以及存储器装满的风险。只要存储器未装满，就不会从队列丢失包，所以从长期的观点看牺牲较忙通道上的数据是有益的，不仅有益于较小队列的传输，而且有益于整个系统。

图 4说明ATDM开关系统最高层体系统结构。主要元件是一个16输入×160位的串行到并行变换器100，分别用于存储和每一条开关输出线相关联想的先进／先出队列中数据包分开的一半的 2块256×160位的随机存取存储器(RAM) 101和102，记录有“数据”RAM中每个相应单元在相关队列中下一单元地址的 256×8位RAM103，队列处理器104，16输出×160 位并行到串行变换器105和用于对输入端口循环计数的4位计数器106。

开关数据以160M bit／s速率、每个包320位的包形式到达16条串行输出线 107。在每个包前端的标题块包含指出该包是否有效或加以忽略的1位，，包含的另4位指出该包应选择到16条开关输出线路中的哪一条的路由。在相位对齐电路 108中将这些包的相位对齐，然后送到串行到并行变换器100，该变换器由32×160位移位寄存器组成，每个输入对应 2个寄存器。对16条输入线路中的每一条，将每个包的前一半(称作“A”一半) 同时装入前16个移位寄存器以便在960ns以后这些寄存器是满的，而将包的后一半(称作 “B”一半) 送入串行到并行变换器的输入侧。在将上述后一半的包装入后16个移位寄存器所需的 960ns期间，16个“A”一半由计数器106选择并行地传到“A 数

据”RAM 101。同样地，在下一组 16个包的前一半装入其移位寄存器时，“B”一半可传送到 “B数据”RAM 102。由于选择每个包传送到合适的RAM块中，队列处理器104使用和该包关联的标题信息用以确定将之写入哪一个实际的 RAM地址。在下一串详细说明队列处理器的操作。

将数据从 RAM移至输入端口的操作序列与上相似只是次序相反。在处理器计算每个输入的“B”包一半的“B”RAM写地址的 16×60ns周期内，该该处理器还通过依次选择各输出队列(例如计数器 106的值) 并选择该队列的尾单元来产生 “A” RAM读地址。类似地，在将“A”包一半写入 “A”RAM的周期内，产生“B”RAM读地址。从这些读地址将数据传送到并行到串行变换器 105如下。当一次一个地、并行地将“B”包一半从“B” RAM移到并行到串行变换器内的第二组的16个移位寄存器时，同时将 “A”一半(已被传送) 从第一组的16个移位寄存器中的每一个移位到16条串行输出线 109，以及同样地，当下一组16个“A”一半从“A” RAM传送时，将 “B”一半的位移出。如果指定输出端口当时没有排队数据，则无须将 RAM内数据传送出，将空包装入应接收到数据的移位寄存器，对标题域加以修改以标识该包为无效的。图 5示出上述数据传送的时序和同步。

这一级的开关体系结构更详细地说明于英国专利 8823493.5号中。

队列处理器

图6示出的队列处理器包括以下元件:

1块16×9位可逆计数器 200(用于存储每个队列单元个数) ，该计数器产生的输出是一个计数器的值(由 4位输入选择) 以及指出该值是否为0的单个位; 记录“自由表”中单元个数并产生指出该值为0时的单个位的单个的 9位可逆计数器 201; 4个16×8位字的寄存器列 202－205，它们分别用于记录每个队列首和尾单元的 RAM地址、各个

“A”包一半所用的RAM读和写地址（以便在“B”包一半时重复使用）；5个8位寄存器206－210，分别用于记录“自由表”首和尾单元的地址、“A数据”RAM读和写地址以及“指针”RAM写地址；用于记录其队列当前最长的输出端口号和当前从RAM装入的输出端口号的2个4位寄存器211和212；记录最长队列单元个数的一个9位寄存器213；分别保持如当前输出端口无队列时被置位的标志、如当前输入包请求的输出端口无队列时被置位的标志以及当存储器满被置位的标志的三个单一位锁存器214－216；1列16×1位的标志217，每个输出端口对应一个标志，当那些端口没有任何排队的包时置位这些标志；用于确定计数器200块中一个数值超过9位寄存器213的9位比较器218；2个2输入×4位的多路转换器219和220和5个2输入×8位的多路转换器221－225，以及最后的组合逻辑块226，用于确定是否进行及何时进行上述元件之间的数据传送。在数据清单A中给出图6中的输入、输出及元件的小结。

如上一节所述，从串行到并行变换器100内的移位寄存器中一次一个地选出数据包，以传送到两块RAM 101和102中的一个。选择每一包时，该包相关的标题信息对队列处理器104变成可利用的。该信息是单个位和4位字的形式，其中所述单个位（图6A中标以“No_Traffic”）为高时，指出该包无效应予忽略，而所述4位数（标为“Req”）说明该包应送到哪一个输出端口（以及队列）。队列处理器的仅有的其它输入是说明哪一个输出端当前正从RAM中取数据的计数器106（“端口”）的4位输出，以及“指针”RAM 103的8位的输出（“Pout”）。每60ns就有一新的包标题到达处理器，而在此期间处理器必须产生两个可能地址对中的一对。所述两对地址是“A数据”RAM写地址和“B数据”RAM读地址或“A数据”RAM读地址和“B数据”RAM写地址。决定产生哪一对取决于每960ns改变状态的时钟线路

(“Wr_A”)，该时钟和串行到并行及并行到串行变换器100和105相同步，使“A”和“B”包一半的读和写如前节所述那样进行(见图5) 。将一个地址对的计算划分为10个6ns的步骤，当10位“走步1”序列发生器(装在“TIMING”块 226中) 相应的输出为高时执行每一步骤。附录 A给出在两个“Wr_A”阶段中的每一段中所执行步骤的算法说明。下面是图 6所示结构如何在多种具体场合下执行这些步骤的详细描述。

来自“TIMING”块226的输出，在图6B中标以1到30，下文称作T1到T30，是用数据清单B给出的布尔式根据标题域的“No-Traffic”位、锁存器214、215和216和比较器218的输出以及10位的“走步1”序列发生器产生的。

情形1:

用存储器中可用的自由空间和有效的输入包产生“A数据”RAM写地址和“B数据”RAM读地址。

由于“自由表”中至少保留有一个自由单元，9位计数器201的输出为低，由于当前处理的输入的包“Port”是有效的，输入线路“No-Traffic”(来自包标题) 为低，这就是图4中计数器 106的输出，它记录当前正在等待“B”包一半的输出端口号，而数据“Req”(来自包标题) 包含输出端口与以及输入的 “A”包一半想要送去的队列。用数值1000000000启动“TIMING”块226内“走步1”时序发生器并启动步骤1。启动寄存器列 204的读操作，以获得将包的“A”一半读出并送到现在正等待相应 “B”一半的输出端口上的“Wr_A”前一阶段所用的 RAM地址。该寄存器列的输出产生“B数据”RAM 102的读地址，一旦稳定，将引起前述“B”包一半到并行到串行变换器105的输入端的传输。“走步 1”序列发生器现在改为数值0100000000从而启动步骤 2。这是一个空步骤，以使改变的输出值得以稳定。在步骤 3，由T13设置多路转换器224以将寄存器206的输出(记录“自由表”RAM首

地址）选作“指针” RAM的读地址。该操作使“自由表”在其首部以后的相邻单元的地址在输入线“Pout”上变成有效的。使寄存器 206中的值写入“Pout”所选单元的寄存器列 205中的 T18变高。由于是“A数据”RAM的写地址，因此必须加以保留以便在“Wr_A”的下个阶段处理输入包的“B”一半时重复使用。用T11设置多路转换器 219，以便将数值“Req”用来从块200中选择一计数器。所选的具体计数器记录由输入的 “A”包一半所请求的输出端口的队列中的单元个数，指出该值是否为 0的标志当驱动T2为高时传送到锁存器 215。驱动T5为高，以便将计数器 201的输出（由于“自由表”中仍有单元，该输出为0）装入锁存器 216。驱动T21为高并将寄存器 216中的数值装入寄存器209，依次，产生“A数据”RAM 101的写地址以及“指针”RAM 103的输入信号。驱动 T20为高，而由T13设置多路转换器 225，以便将寄存器列203的输出（输入包请求的输出端口的队列尾端单元的RAM地址）装入依次驱动“指针”RAM 103的写地址的寄存器210。在步骤4，驱动“A数据” RAM 101的写允许线路（T27）为高使输入的“A”包一半写到由寄存器 209所载的地址中去，这样填入“自由表”首部单元。T4置为高使计数器201（自由RAM单元号）递减。在步骤 5，由T11设置多路转换器219以便将数值“Req”用作寄存器列 202 的写地址。它选择包含由输入包请求的队列首部单元地址的寄存器。由 T11设置多路转换器223，以使寄存器 206中的数值（当前“A数据” RAM写地址）可写入寄存器列202。如锁存器215为高，即意味着输入包请求的输出端口当前不存在队列， 那么驱动寄存器列 202 的写允许线（T16）为高，而包含新包的RAM单无变成所请求端口的队列之首。驱动寄存器列 203（包含每个队列尾单元的 RAM 地址） 的写允许线路（T17）以便也写入寄存器206中的数值。这使得所请求端口队列尾端的地址指出该新包。驱动T7为高使块 200中所请求端口队列中单元个

数递增。步骤6是另一空步骤。在步骤 7，分别由T11和T0设置多路转换器219和220，以便将寄存器 211中的数值(其队列为当前最长的输出端口号)用于从块 200中选择相应计数器。驱动T9为高将该计数器数值传送到记录最长队列长度的寄存器 213以使之成为最新的。在步骤8，分别由T14和T13设置多路转换器221和222，而当T15变高时将输入线路“Pout”的值装入寄存器 206。该数据由步骤3启动的“指针”RAM 103 读操作所产生，该值是“自由表”旧的首部以后的下一单元的RAM地址。而现在成为“自由表”的新的首部的 RAM地址。T24变为高使输入线路“Pout”上的数值装入寄存器 212，选择由并行到串行变换器105中的哪一个移位寄存器接收从“B”RAM 102中读出的“B”包一半。在步骤9，假设锁存器215中的数值为0(指出输入的“A”包一半请求的输出端口已有对列存在)，那么驱动“指针”RAM 103 的写允许线路(T25)为高而将寄存器 209中的数值(新包RAM的地址)写入寄存器 210记录的地址(现以包含新包的队列的老的“尾”地址)中去。该动作有效地将一新的 “A”包一半“链接”到其队列的端部。由T11设置多路转换器 219以便从块200中选出记录有新包队列单元个数的计数器。用比较器218将该值与寄存器213中的值(当前最长队列的长度)进行比较。如果比较器入输出为高，那么新包队列的长度会超过前最长队列的长度，T2 变为高，而将数值“Req”装入记录其队列最长的端口号的寄存 211，而现在指出由新包请求的端口队列。如果寄存器列 217 的输出(指出是否有 “A” 包一半队列，并在前一“Wr_A”阶段写到当前输出端口)为0，那么T29变为高，并将在并行到串行变换器 105的输入端等待的“B”包一半装入在步骤8选择的输出移位寄存器中，否则 T28变为高，设置该移位寄存器，使之包含其标题域的“No_Traffic”位具有标志为无效包的1 的数值的空包。最后，在步骤 10(“TIMING”块中“走步1”序列发生器的值为

0000000001），驱动T30为高并递增端口计数器106，从而选择串行到并行变换器 100中包含下一待处理包的移位寄存器。

情形2：

存储器已满并有一输入的有效包时产生“A数据” RAM 写地址和“B数据”RAM读地址。

如前一种情形那样，从包标题输入的“No_Traffic”为低，数值“Port”为当前正等待“B”包一半的输出端口号，而数值“Req”为输出端口号以及输入的 “A”包一半希望送入的队列。然而，计数器201 的输出这时为高，指出不存在“自由表”，因此该存储器已满。处理器的操作除了以下差别，如情形1中那样进行。在步骤1，分别由T11和T10设置多路转换器 219和220以便用寄存器211的数值来选择包含最长队列首部单元的RAM地址的寄存器列202中的单元。分别由 T14和T13设置多路转换器 221和 222，驱动T15为高使该地址装入寄存器206（ 通常记录“自由表”首部单元的 RAM地址）。多路转换器219和220的设置也使用寄存器 211从块200中选出记录最长对列单元个数的计数器。驱动T8为高，并递减该计数器中的值。在步骤 3，当驱动T5为高时，从计数器201送到锁存器216的数值为 1，指示不存在“自由表”。从寄存器206传送到寄存器209的数值这样变成“A 数据” RAM 101的写地址，而“指针” RAM 103 的输入现在成为最长队列首部单元的地址而不是“自由表”首部单元的地址。将寄存器 206选作“指针 ” RAM 的读地址使最长队列首部以后邻接的单元地址在输入线路“Pout”上变成有效的。在步骤 4，当驱动“A数据”RAM写允许线路T27为高时，输入的 “A”包一半重写最长队列首存储的“A”包一半。T4保持低，而计数器201的数值（ “自由表”中单元数，即0）不递减。在步骤7，分别由T11和T10设置多路转换器219和220，使寄存器211中的数值用来选择寄存器列202中包含最长队列首单元RAM地址的单元。

用由T11设置的多路转换器 223驱动T16为高使该单元为“指针”RAM 103的输出由 “Pout”所改写。这样做的效果是使最长队列的首地址指向该队列前“首”单元(刚刚由输入包所重写) 后所邻接 RAM单元。在步骤 8，仍然如情形 1那样将“Pout”装入寄存器 206(记录“自由表”首单元的 RAM地址) ，只是由于没有“自由表”，该动作无影响。

情形3:

出现有效的输入包时产生“B数据”RAM

写地址和“A数据”RAM读地址。

来自包标题的输入信号“No_Traffic为低，数值“Pout”这时是等待“A”包一半的输出端口号而数值“Req”未使用。在步骤 1，由T11和T10分别设置多路转换器 219和 220，以便将数值“Port”用作寄存器列 202的读地址，用以选出当前输出端口队列首单元的地址。驱动 T19为高，将该地址装入产生“A数据”RAM 101读地址的寄存器208。多路转换器 219和220的设置也产生用于从块 200中选出记录当前输出端口队列大小的计数器的数值“Port”。驱动T1为高，将指出该计数器值是否为 0的标志装入锁存器 214。驱动“B数据”RAM写允许线路 T26为高并将输入的“B”包一半写到由寄存器列205给出地址的“B”RAM 102中。该地址与在前 “Wr_A”阶段中将相应“A”包一半写入“A”RAM101时所用的地址相同。步骤2为空步骤。在步骤 3，T22变高，使寄存器208中数值写入寄存器列 204中由“Port”所选单元。由于是“A数据”RAM 读地址因此须预保存以便在“Wr_A” 下一阶段处理输出包的“B”一半时重复使用。由T13设置多路转换器 224以便将寄存器 208 的输出(记录当前输出端口队列首的RAM地址) 选作“指针”RAM 的读地址。该动作使输出包队列的下一单元的地址变成对输入线路 “Pout”是有效的。驱动 T21 为高，而依次将寄存器

208中数值装入产生“指针”RAM103的输入的寄存器209中。驱动T5为高使计数器 201的输出装入锁存器216。驱动T20为高并由 T13设置多路转换器225使寄存器207中的数值(“自由表”尾的 RAM地址) 装入依次驱动“指针”RAM 103 写地址的寄存器 210。开始“A数据”RAM 101的读操作使寄存器 208记录的RAM地址中的 “A”包一半传送到并行到串行变换器105的输入端。由T11和 T10分别设置多路转换器 219和220使“Port”值从块200中选出对应于当前输出端口的计数器。如果锁存器214的输出为0(意味着当前输出端口至少有一个包在队列中并已经读出) , 那么驱动T8为高并递减该计数器。驱动T6为高并将锁存器214中的数值存储在寄存器列217(由当前输出端口号所选址) 中以便在处理输出包的相应“B”一半时进行校验。在步骤4, 除非锁存器 214输出为 0什么也不做。在这种情形下, 驱动 T23为高使寄存器208中数值装入寄存器207, 该寄存器记录“自由表”尾端单元的 RAM地址而现在指出被输出的 “A”包一半空出的单元。驱动T3为高, 使记录自由 RAM单元201的个数的计数器递增。如锁存器216的输出为高(意味着没有“自由表”, 因此空出的单元是存储器中唯一的自由单元) , 那么用由 T13设置的多路转换器222驱动T15为高, 以便将寄存器208中的数值装入寄存器 206。该数值保持“自由表”首单元的RAM地址而现在还指出由输出“A”包一半空出的单元。在步骤5, 分别用T11和T10设置多路转换器219和220, 以便将寄存器21中的数值(其队列当前最长的输出端口号) 用来从块 200中选出相应的计数器, 驱动T9为高并将该计数器数值传送到记录最长队列长度的寄存器 213从而加以刷新。步骤6为空步骤。在步骤7, 如果锁存器214的输出为0(意味着当前输出端口曾有排队的包所以已经读出) , 那么设置多路转换器219和220, 使数值“Port”用来选择寄存器列 102中包含当前输出端口队列首的RAM地址的单元。然后(再设锁存器214为0) 由T11来

设置多路转换器223，将T16驱动为高，使所选单元被“指针”RAM103的输出“Pout”所重写。这样做的效果是使当前输出端口的队列首地址指向该队列在前“首”单元(刚刚被读出)以后的下一单元的RAM地址。在步骤8，驱动T24为高使输入线路“Port”上的数值装入寄存器212，在并行到串行变换器105中选择移位寄存接收从“A”RAM 101中读出的“A”包一半。在步骤9，如果锁存器器216的输出为0(意味着在读出当前“A”包一半之前存储器中至少有一自由单元)，那么驱动“指针”RAM 103的写允许线路为高，保存于寄存器210中的写地址指出当前“自由表”尾单元，而保存在寄存器209中的输入是刚刚空出的“A数据”RAM单元的地址。这样空出单元被“链接”到“自由表”末端。在步骤10，如果锁存器214的输出为0(指出当前输出端口有排队包并已读出)，那么T29变为高，而将在并行到串行变换器105的输入端等待的“A”包一半装入步骤8所选输出移位寄存器，否则T28变为高，而设置该移位寄存器以包含其标题域“No_Traffic”位具有标志其无效的1值的空包。

数据清单A

图6示出的输入、输出和寄存器的说明

A_DrdAdd: 给出A数据RAM[101]的读地址的8位输出总线

A_Dwe: 允许对A数据RAM[101]写的1位输出线路

A_DwrAdd: 给出A数据RAM[101]的写地址的8位输出总线

B_DrdAdd: 给出B数据RAM[102]的读地址的8位输出总线

B_Dwe: 允许对B数据RAM[102]写的1位输出线路

B_DwrAdd: 给出B数据RAM[102]的写地址的8位输出总线

B_PortEmpty(217):16×1位标志，用于对每个端口记录“PortEmpty”

数值以便对数据 RAM的B一半存取时重复使用这些端口

B_RdReg(204) : 16×8位寄存器列，对每个端口记录保持“RdReg”数据以便在读出包的B一半时重复使用这些端口

B_WrReg(205) : 16×8位寄存器列，对每个端口记录数据RAM写地址以便在写包的B一半时重复使用这些端口

Ctr(200) :16×9位计数器指出每个端口队列当前包个数。可通过 4位地址线选择给定的计数器进行增／减和(或) 读操作。如其为0时输出计数器值和指示标志。

FlEmpty(216) : 保存“FlpCtr”输出的一位标志。

FlpCtr(201) : 记录自由表当前单元个数的 9位位计数器。该输出为指出该计数器是否为0的标志。

FlpHead(206) : 记录自由表首RAM地址的8位寄存器。

FlpTail(207) : 记录自由表尾RAM地址的8位寄存器。

Head(202) : 对每个端口记录端口表首的 RAM地址的 16×8位寄存器列。

Load: 使数据RAM的输出存储到所选输出锁存器的1位输出线路。

MaxCtr(213) : 记录“MaxPort”的“Ctr”数值的9位寄存器。

MaxPort(211) : 指出哪一个端口为最长表的4位寄存器。

NewReg(209) : 记录A数据RAM写地址和指针RAM(103) 输入数据的8位寄存器。

Next_Port: 选择下一“Port”的1位输出线路。

No_Traffic: 来处输入包标题域的、指出该包是否含有效数据的1位。

Null: 设置所选输出锁存器的标题域中No_Traffic 位的1

位输出线路。

OP_Select: 选择“Null”的输出锁存器或“Load”操作的4位输出总线。

Pin: 到指针RAM(103)的8位输入。

Port: 选择当前输入/输出端口的4位输入。

Port2(212): 保存“Port”数值的4位寄存器。

PortEmpty(214): 当前“Port”的“Ctr”值为0时置位的1位标志。

Pout: 指针RAM(103)的8位输出

PrdAdd: 记录指针RAM(103)读地址的8位输出总线

Pwe: 允许对指针RAM(103)写的1位输出线路

PwrAdd(210): 包含指针RAM(103)写地址的8位寄存器和输出总线。

RdReg(208): 记录当前正读出包的RAM地址的8位寄存器。

Req: 输入包标题域中指出包指定输出端口以及包应存储的表的4位。

ReqEmpty(215): 当“Req”端口的“Ctr”数值为0时置位的1位标志。

Tail(203): 对每个端口记录该端口表尾的RAM地址的16×8位寄存器列。

Wr_A: 1位输出线路，用于确定该处理器是否时A数据RAM写入和从B数据RAM中读出，或对B数据RAM写入和从A数据RAM中读出。

数据清单 B

TIMING块输出(见图6)

P0…P9对应于“走步1”序列发生器用于标识10个连续的6ns时钟周期的输出

注：给出写允许线路为有效高电平

1) PortEmpty_W=not(Wr_A) AND (P0 OR P1)

2) ReqEmpty_W=Wr_A AND (P2 OR P3)

3) FlpCrt_Inc=not (Wr_A) AND (P3 OR P4) AND not(PortEmpty)

4) FlpCtr_Dec=Wr_A AND (P3 OR P4) AND not (No_Traffic) AND not (FlEmpty)

5) FlEmpty_W=P2

6) PortEmptyB_W =not(Wr_A) AND (P2 OR P3)

7) Ctr_inc=Wr_A AND (P4 OR P5)AND not(No_Traffic)

8) Crt_Dec=(Wr_A AND (P0 OR P1) AND not (No_Traffic) AND (FlpCtr=0)) OR (not (Wr_A) AND (P2 OR P3) AND not (PortEmpty))

9) MaxCrt_W=(Wr_A AND (P6 OR P7)) OR (not (Wr_A) AND (P4 OR P5))

10) Ctr_Mux=not (Wr_A) AND (P0 OR P1 OR P2 OR P3 OR P6 OR P7)

11) Head_Mux=not (Wr_A) OR P0 OR P1 OR P6 OR P7

12) MaxPort_W=Wr_A AND P8 AND (Crt[]>MaxCtr)

13) Wr_A

14) FlpHead_Mux=P7

15) FlpHead_W=(Wr_A AND (((P0 OR P1) AND (FlpCtr=0)) OR (P7 AND not (No_Traffic)))) or (not (Wr_A) AND P3 AND not (PortEmpty) AND FlEmpty)

16) Head_W=(Wr_A AND not (No_Traffic) AND (((P4 OR P5)

```
        AND ReqEmpty) OR ((P6 OR P7) AND FlEmpty))) OR
        (not (Wr_A) AND (P6 OR P7) AND not (PortEmpty))
17) Tail_W=Wr_A AND (P4 OR P5) AND not (No_Traffic)
18) WrRegB_W=Wr_A AND (P2 OR P3)
19) RdReg_W=not (Wr_A) AND (P0 OR P1)
20) Pw Add_W=P2 OR (Wr_A AND P3)
21) NewReg_W=P2 OR (Wr_A AND P3)
22) RdRegB_W=not (Wr_A) AND (P2 OR P3)
23) FlpTail_W=not (Wr_A) AND P3 AND not (PortEmpty)
24) Port2_W=P7
25) Pwe =(P7 OR P8 OR P9) AND ((Wr_A AND not (ReqEmpty)) OR
      (not (Wr_A) AND FlEmpty))
26) B_Dwe =not (Wr_A) AND (P2 OR P3 OR P4 OR P5 OR P6 OR P7
      OR P8 OR P9) AND not (No_Traffic)
27) A_Dwe =Wr_A AND (P3 OR P4 OR P5 OR P6 OR P7 OR P8 OR P9)
      AND not (No_Traffic)
28) Null =(Wr_A AND P8 AND PortEmpty_B[])) OR (not(Wr_A) AND
      P9 AND PortEmpty)
29) Load =(Wr_A AND P8 AND not (PortEmpty_B[])) OR (not (Wr_
      A) AND P9 AND not (PortEmpty))
30) Next_Port =P9
```

本清单包括对前一页算法带标号的语句的简短注释。

A数据RAM写／B数据RAM读(Wr_A)

1) 如果自由表为空(即，队列已满)，则准备改写最长表表首。

2) 从该端口前A数据RAM读的同样地址启动B数据RAM读操作。

3) 如最长队列被改写，那么递减适当通道的计数。

4）启动指针RAM读操作，以获得自由表中下一单元或最长表中的下一单元。

5）复制A数据RAM的写入地址以便在写入B数据RAM时可重复使用。

6）设置请求表为空的标志（由于在语句19中测试了该条件并由Ctr[Req]可能已由语句14所改变因此是必须的）。

7）设置自由表为空的标志（由于在语句15已测试该条件并且FlpCtr可能已被语句11所改变，因此是必须的。

8）锁存写入指针RAM所用的数值（由于FlpHead可能被语句17所修改而在语句19之前不能开始写操作因此是必须的）。

9）锁存写入指针RAM时所用的地值（由于Tail(Req)可能被语句13所修改而在语句19之前不能开始写操作因此是必须的）。

10）如存在传输则将新包的A一半写入A数据RAM。写地址是自由表之首，或者是当自由表为空时，该写地址是最长表之首。

11）如存在传输而且自由表未空，则递减自由表计数。

12）如果请求通道不存在表，那么通过标记其首部指出新包而开始一个新表。

13）如果有传输，那么使请求通道的尾指针指向一新包……。

14）……递增该通道的计数。

15）如最长表已被重写，那么将该表首指针移下一个位置。

16）保持最长表计数为最新的。

17）如果有传输，便将自由表首移到该表下一位置（如该自由表为空，那么不动作）。

18）锁存当前端口的数值（考虑到A数据RAM读周期的连续性）。

19）如果有请求信道的表，那么将新的包链接到该表末端。

20）如果请求信道的表比最长表要长，则使之为新的最长的表。

21）如在语句2读出的B RAM单元中存在有效数据，那么将B数据RAM的

输出装入输出寄存器……

22）……否则设置输出寄存器标题域中以No_Traffic位。

23）移到下一端口。

A数据RAM读／B数据RAM写(not(Wr_A))

1）准备读出该端口的表首。

2）设置该端口表为空的标志(因为在语句 12、13、14、16、19和20中已测试该条件而且Ctr(Port)可能被语句10修改，所以是必须的）。

3）如存在传输(B一半）则启动B数据RAM的写操作，写入地址和该端口前一次A数据RAM写入地址相同。

4）复制从 A数据RAM读出时所用地址以便在从B数据RAM读出时被重复使用。

5）锁存在写入指针RAM时所用数值(由天在完成该写操作之前，RdReg被下一语句1所修改因此是必须的）。

6）设置自由表为空的标志(由于语句18已对该条件测试并且FlpCtr可能已被语句14所修改因此是必须的）。

7）锁存在写入指针 RAM时所用地址(由于FlpTail可由语句13所修改并且在语句13之前不能开始写操作因此是必须的）。

8）启动从A数据RAM的读操作。

9）启动指针RAM读操作以获得该端口表的下一项。

10）如果存在该端口的读出数据，则递减适当的计数。

11）存储确定是否从A RAM读出有效数据的标志以便为相应的B RAM读操作所重复使用。

12）如已有单元空出并且不存在自由表，则通过使该表首指向该空出单元而启动新的自由表。

13）如已经空出一单元，则使该单元为空表之尾……

14）……递增自由表计数器。

15）保持最长队列的计数为最新的。
16）如已经读出有效包，则将其表的首指针移下一个位置。
17）锁存当前端口的数值（由于该数值在用于装载合适的输出寄存器的同时已经递增（语句19、20和21），因此是必须的）。
18）如存在自由表，则将新空出的单元链接到该表的端点上。
19）如果A数据RAM已产生有效数据（即该端口的表未空）则将该数据装入输出寄存器……
20）……否则设置输出寄存器标题中的No_Traffic位。
21）移到下一端口。

这是由图6所示队列处理器所执行的算法。已经划分为“Wr_A”的两个分开的阶段，使得在“Wr_A”为高的第一阶段，该算法产生A数据RAM写地址和B RAM读地址，而在“Wr_A”为低的第二阶段，产生A数据RAM读地址和B数据RAM写地址。将每个阶段进一步划分为每个持续6ns的10个独立的步骤，以便可同时执行给定步骤中所有语句。

```
if Wr_A then
begin

(*_____0ns ___*)

1) if (FlpCtr=0) then FlpHead:=Head[MaxPort];
2) drd_b(RdReg_B[Port_B]);
3) if not(No_Traffic) and (FlpCtr=0)then dec(Ctr[MaxPort]);
(*_____ 6ns ___*)
(*_____ 12ns ___*)
```

```
4)  prd(FlpHead);
5)  WrReg_B[Port]: =FlpHead;
6)  ReqEmpty: =(Ctr[Req] =0);
7)  FlEmpty: =(FlpCtr =0);
8)  NewReg: =FlpHead;
9)  PwrAdd: =Tail[Req];

(*_____ 18ns ___*)
10) if not(No_Traffic) then dwr_a(NewReg);
11) if not(No_Traffic) and not (FlEmpty) then dec (FlpCtr);

(*_____ 24ns ___*)
12) if not(No_Traffic) and ReqEmpty then Head[Req]: =FlpHead:
    if not(No_Traffic) then
    begin
13)   Tail[Req]: =FlpHead;
14)   inc(Ctr[Req])
    end;

(*_____ 30ns ___*)
(*_____ 36ns ___*)
15) if not(No_Traffic) and FlEmpty then Head[MaxPort]: =Pout;
16) MaxCtr: =Ctr[MaxPort];
(*_____ 42ns ___*)
```

```
17) if not(No_Traffic) then FlpHead:=Pout;
18) Port2:=Port;
(*_____ 48ns ___*)
19) if not(ReqEmpty) then pwr(PwrAdd NewReg);
20) if (Ctr[Req]>MaxCrt) then MaxPort:=Req;
    if not(PortEmpty_B [Port]) then
21)    load(Port2)
    else
22)     null(Port2);
(*_____ 54ns ___*)

23) inc(Port);


(*_____ 60ns ___*)
end
else
begin
(*_____ 0ns ___*)

1) RdReg:=Head[Port];
2) PortEmpty:=(Ctr[Port]=0);
3) if not(No_Traffic) then dwr_b(WrReg_B[Port]);

(*_____ 6ns ___*)
(*_____ 12ns ___*)
```

```
4) RdReg_B[Port] =RdReg;
5) NewReg: =RdReg;
6) FlEmpty =(FlpCtr =0);
7) PwrAdd: =FlpTail;
8) drd_a(RdReg);
9) prd(RdReg);
10)if not(PortEmpty)then dec(Ctr[Port]);
11)PortEmpty_B [Port]: =PortEmpty;
(*_____ 18ns ___*)
12) if not (PortEmpty)and FlEmpty then FlpHead: =RdReg;
    if not(PortEmpty)then
    begin
13)   FlpTail: =RdReg;
14)   inc(FlpCtr)
    end
(*_____ 24ns ___*)
15) MaxCtr: =Crt[MaxPort];



(*_____ 30ns ___*)
(*_____ 36ns ___*)
16) if not(PortEmpty) then Head[Port]: =Pout;
(*_____ 42ns ___*)
17) Port2: =Port;
(*_____ 48ns ___*)
```

```
18) if not(FlEmpty) then pwr(PwrAdd NewRdg);


(*_____ 54ns ___*)
19) if not(PortEmpty) then
        load(Port2)
    else
20)    null(Port2);
21) inc(Port);
(*_____ 60ns ___*)
end
```

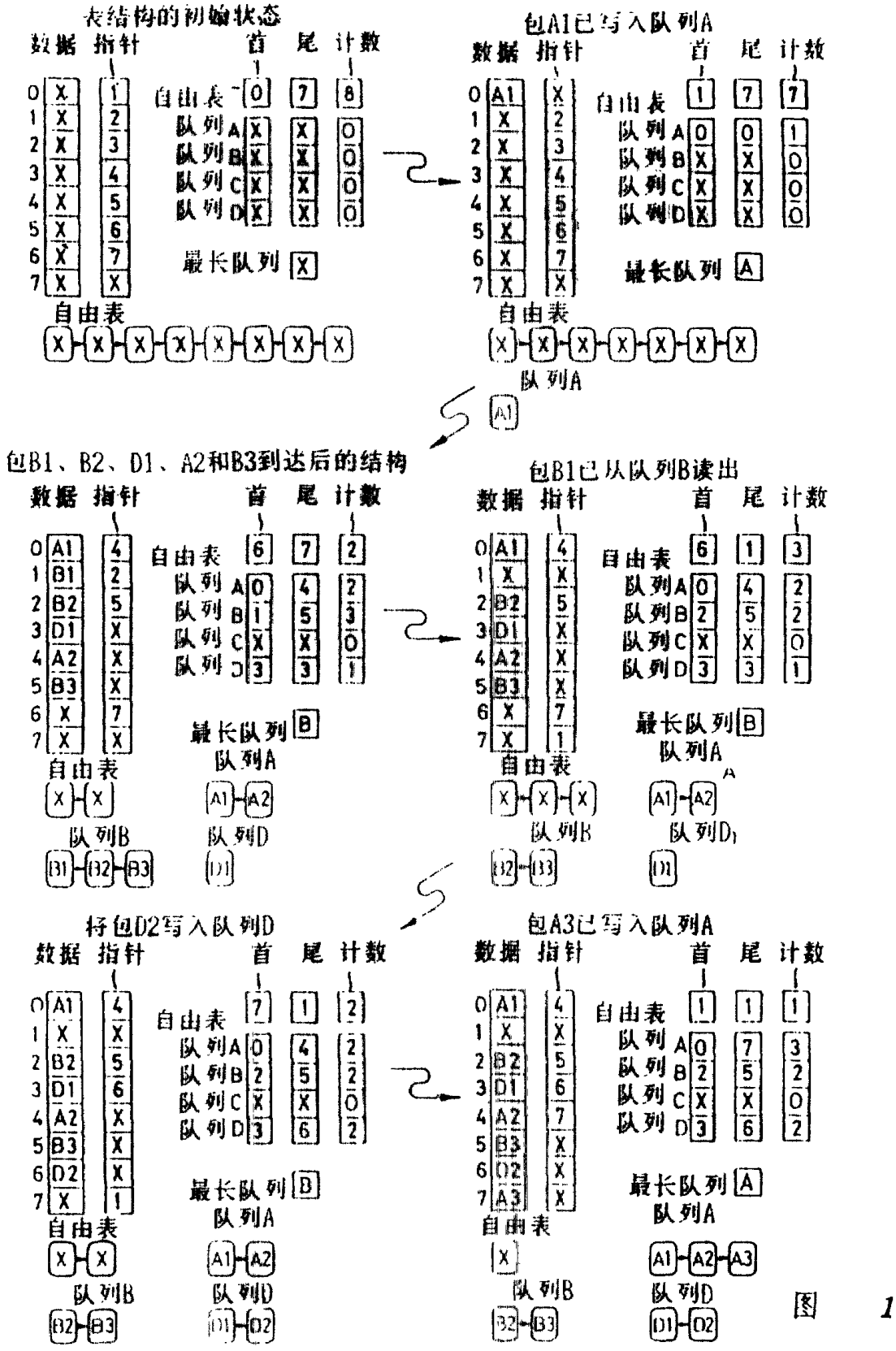

表结构的初始状态
数据 指针
首 尾 计数
自由表
队列A
队列B
队列C
队列D
最长队列
自由表
包A1已写入队列A
包B1、B2、D1、A2和B3到达后的结构
队列B
队列D
包B1已从队列B读出
队列A
将包D2写入队列D
包A3已写入队列A

图　1

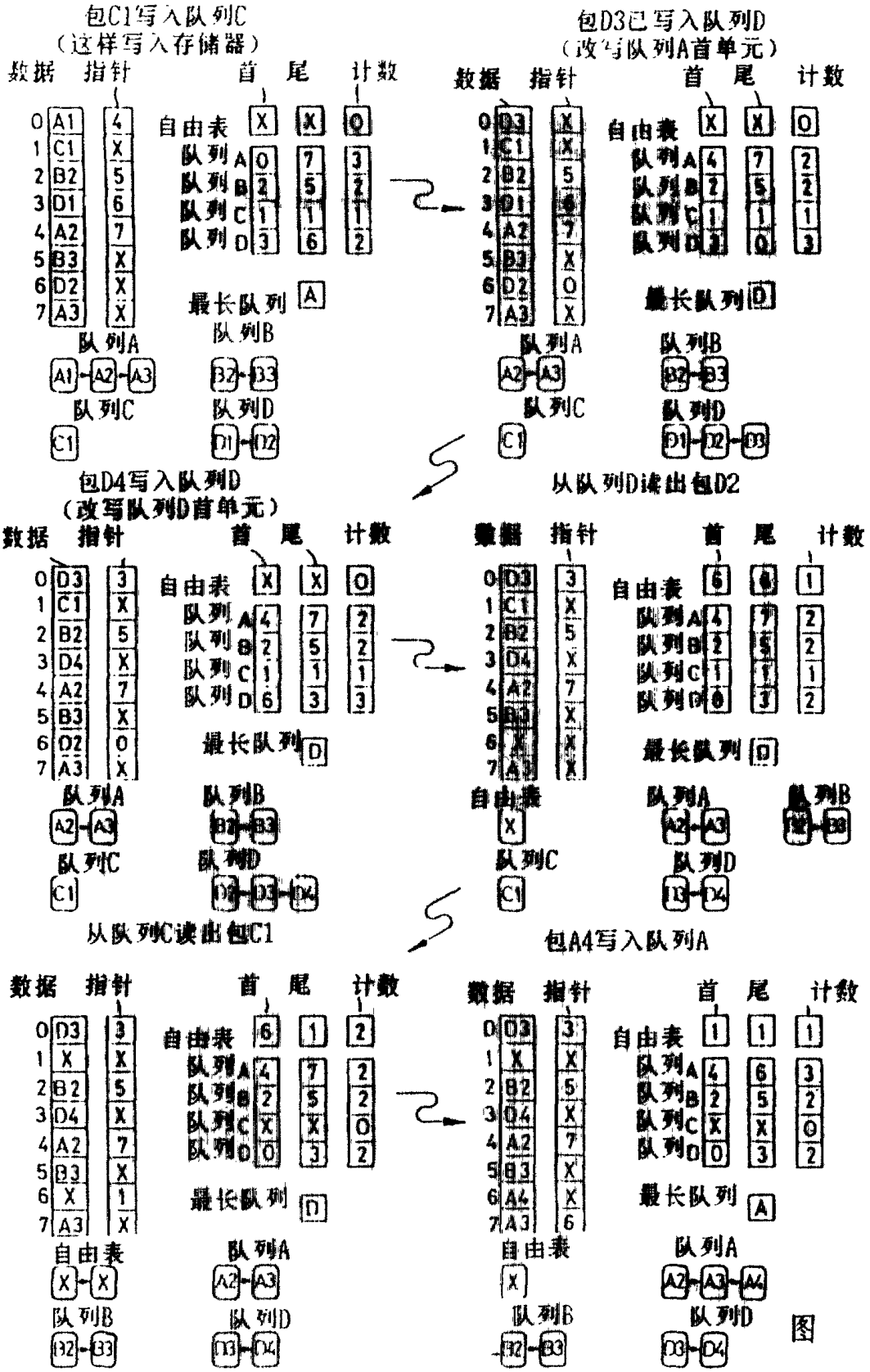

包C1写入队列C
（这样写入存储器）
数据 指针 首 尾 计数
自由表
队列A
队列B
队列C
队列D
最长队列
队列A
队列B
队列C
队列D
包D3已写入队列D
（改写队列A首单元）
包D4写入队列D
（改写队列D首单元）
从队列D读出包D2
从队列C读出包C1
包A4写入队列A

图 1A

图 2

用16×16开关伪随机传输1E6个包的仿真结果
（总队列深度=256）

```
算法1（牺牲输入数据），或2（牺牲最大队列数据）：1
输入总队列深度：256
输入端口个数：16
输入传输密度率.
端口  0:  16
端口  1:  17
端口  2:  18
端口  3:  19
端口  4:  20
端口  5:  21
端口  6:  22
端口  7:  23
端口  8:  24
端口  9:  25
端口 10:  26
端口 11:  27
端口 12:  28
端口 13:  29
端口 14:  30
端口 15:  31
输入总占有率（%）：100
输入仿真包个数：1000000

端口  0:    入=42506,    出=32177,    丢失=10329(24.30%)
端口  1:    入=45256,    出=34275,    丢失=10981(24.26%)
端口  2:    入=47934,    出=36307,    丢失=11627(24.26%)
端口  3:    入=50609,    出=38511,    丢失=12098(24.90%)
端口  4:    入=53077,    出=40233,    丢失=12844(24.20%)
端口  5:    入=56029,    出=42450,    丢失=13579(24.24%)
端口  6:    入=58224,    出=44228,    丢失=13996(24.04%)
端口  7:    入=60918,    出=46369,    丢失=14549(23.88%)
端口  8:    入=64386,    出=48873,    丢失=15513(24.09%)
端口  9:    入=66068,    出=50199,    丢失=15869(24.02%)
端口 10:    入=69169,    出=52413,    丢失=16756(24.22%)
端口 11:    入=71625,    出=54383,    丢失=17241(24.07%)
端口 12:    入=74390,    出=56278,    丢失=18112(24.35%)
端口 13:    入=77358,    出=58510,    丢失=18848(24.36%)
端口 14:    入=79725,    出=60482,    丢失=19243(24.14%)
端口 15:    入=82726,    出=62673,    丢失=20053(24.24%)

总计:       入=1000000,  出=758362,   丢失=241638(24.16%)

最大总队列深度=256
最大单队列深度=16.00
```

图　2A

```
算法1（拒性输入数据），或2（牺牲最长队列中数据）:2
输入总队列深度：256
输入端口个数：16
输入传输速度率...
端口 0:  16
端口 1:  17
端口 2:  18
端口 3:  19
端口 4:  20
端口 5:  21
端口 6:  22
端口 7:  23
端口 8:  24
端口 9:  25
端口10:  26
端口11:  27
端口12:  28
端口13:  29
端口14:  20
端口15:  31
输入总占有率（%）：100
输入仿真包个数：1000000

端口 0:    入 = 42506,    出 = 42506,    丢失 = 0 (0.00%)
端口 1:    入 = 45256,    出 = 45256,    丢失 = 0 (0.00%)
端口 2:    入 = 47934,    出 = 47934,    丢失 = 0 (0.00%)
端口 3:    入 = 50609,    出 = 50609,    丢失 = 0 (0.00%)
端口 4:    入 = 53077,    出 = 53077,    丢失 = 0 (0.00%)
端口 5:    入 = 56029,    出 = 56017,    丢失 = 12 (0.02%)
端口 6:    入 = 58224,    出 = 58127,    丢失 = 97 (0.17%)
端口 7:    入 = 60918,    出 = 60634,    丢失 = 284 (0.47%)
端口 8:    入 = 64386,    出 = 62261,    丢失 = 2125 (3.30%)
端口 9:    入 = 66068,    出 = 62401,    丢失 = 3667 (5.55%)
端口10:    入 = 69169,    出 = 62521,    丢失 = 6648 (9.61%)
端口11:    入 = 71625,    出 = 62531,    丢失 = 9094 (12.70%)
端口12:    入 = 74390,    出 = 62532,    丢失 = 11858 (15.94%)
端口13:    入 = 77358,    出 = 62531,    丢失 = 14827 (19.17%)
端口14:    入 = 79725,    出 = 62536,    丢失 = 17189 (21.56%)
端口15:    入 = 82726,    出 = 62534,    丢失 = 20192 (24.41%)

总计：     入 = 1000000,  出 = 914007,   丢失 = 85993 (8.60%)

最大总队列深度 = 256
最大单队列深度 = 16.00
```

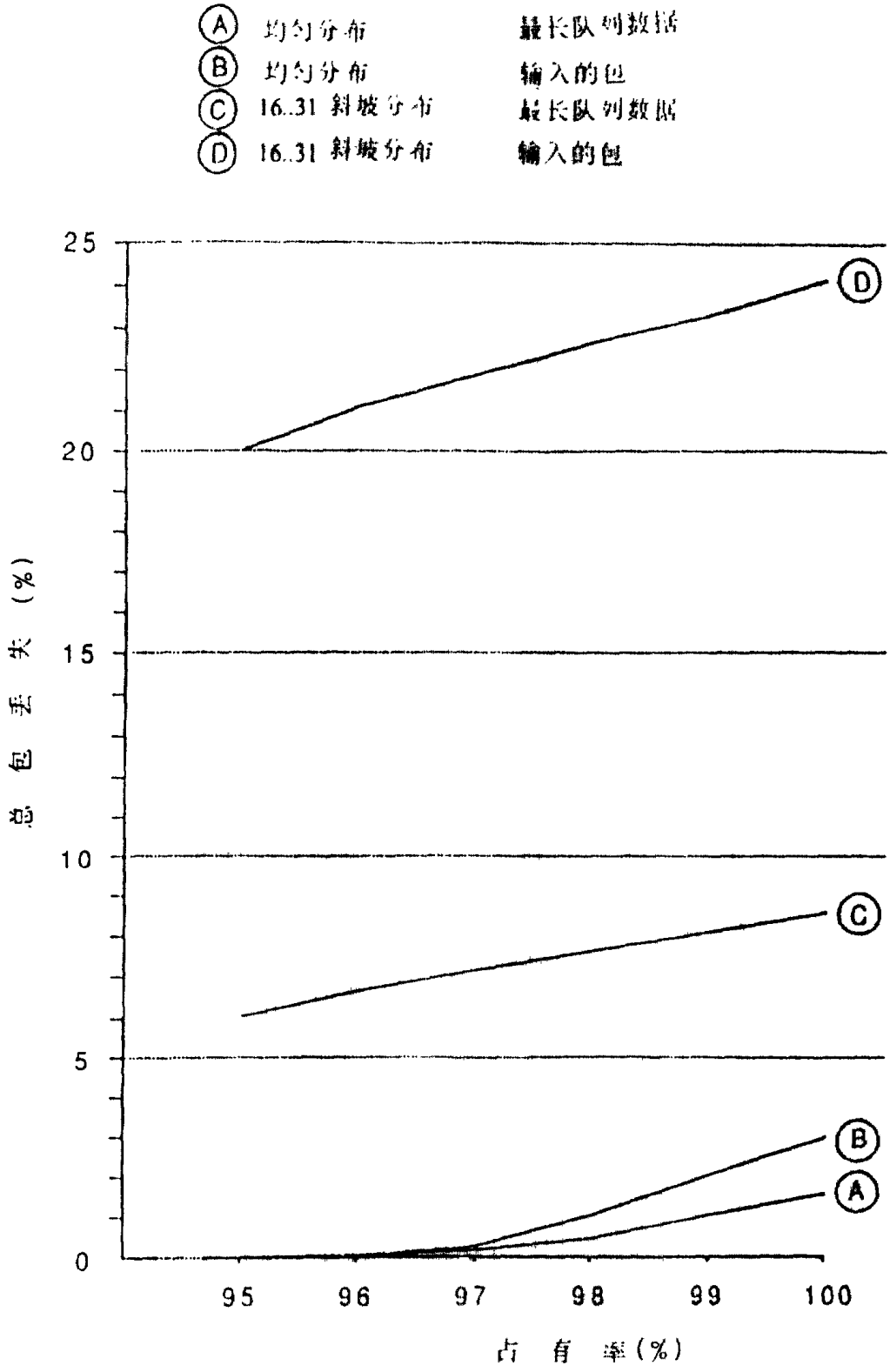

Ⓐ 均匀分布 最长队列数据
Ⓑ 均匀分布 输入的包
Ⓒ 16..31 斜坡分布 最长队列数据
Ⓓ 16..31 斜坡分布 输入的包
总包丢失（%）
25
20
15
10
5
0
95
96
97
98
99
100
占有率（%）

图 3

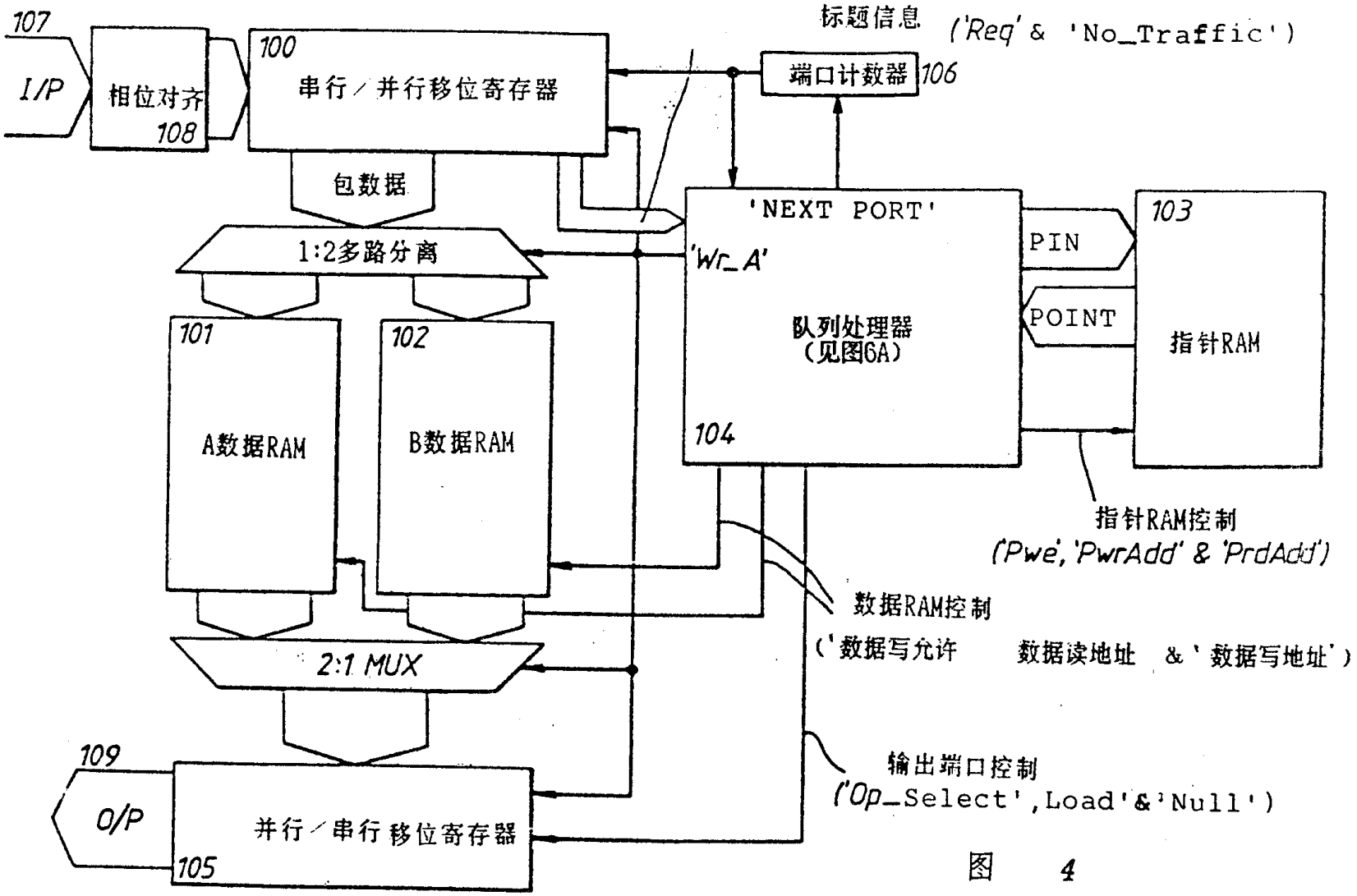

107
I/P
相位对齐
108
100
串行／并行移位寄存器
标题信息
('Req' & 'No_Traffic')
端口计数器
106
包数据
1:2多路分离
'NEXT PORT'
'Wr_A'
队列处理器
(见图6A)
104
PIN
POINT
103
指针RAM
101
A数据RAM
102
B数据RAM
指针RAM控制
('Pwe','PwrAdd' & 'PrdAdd')
数据RAM控制
('数据写允许　　数据读地址　& '数据写地址')
2:1 MUX
109
O/P
并行／串行移位寄存器
105
输出端口控制
('Op_Select',Load'&'Null')

图　4

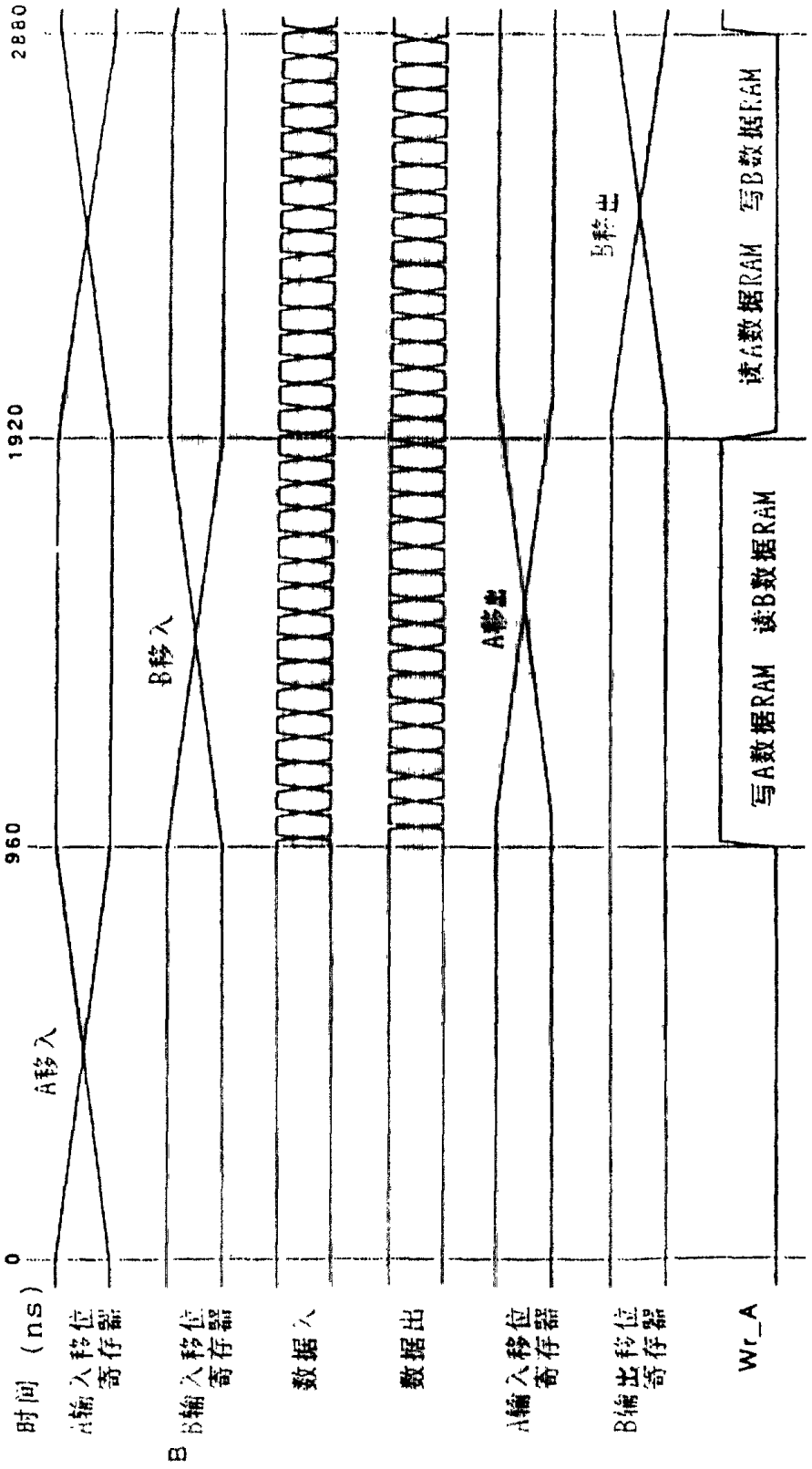

时间 （ns）
0
960
1920
2880
A输入移位寄存器
B输入移位寄存器
数据入
数据出
A输入移位寄存器
B输出移位寄存器
Wr_A
A移入
B移入
A移出
B移出
写A数据RAM 读B数据RAM
读A数据RAM 写B数据RAM

图 5

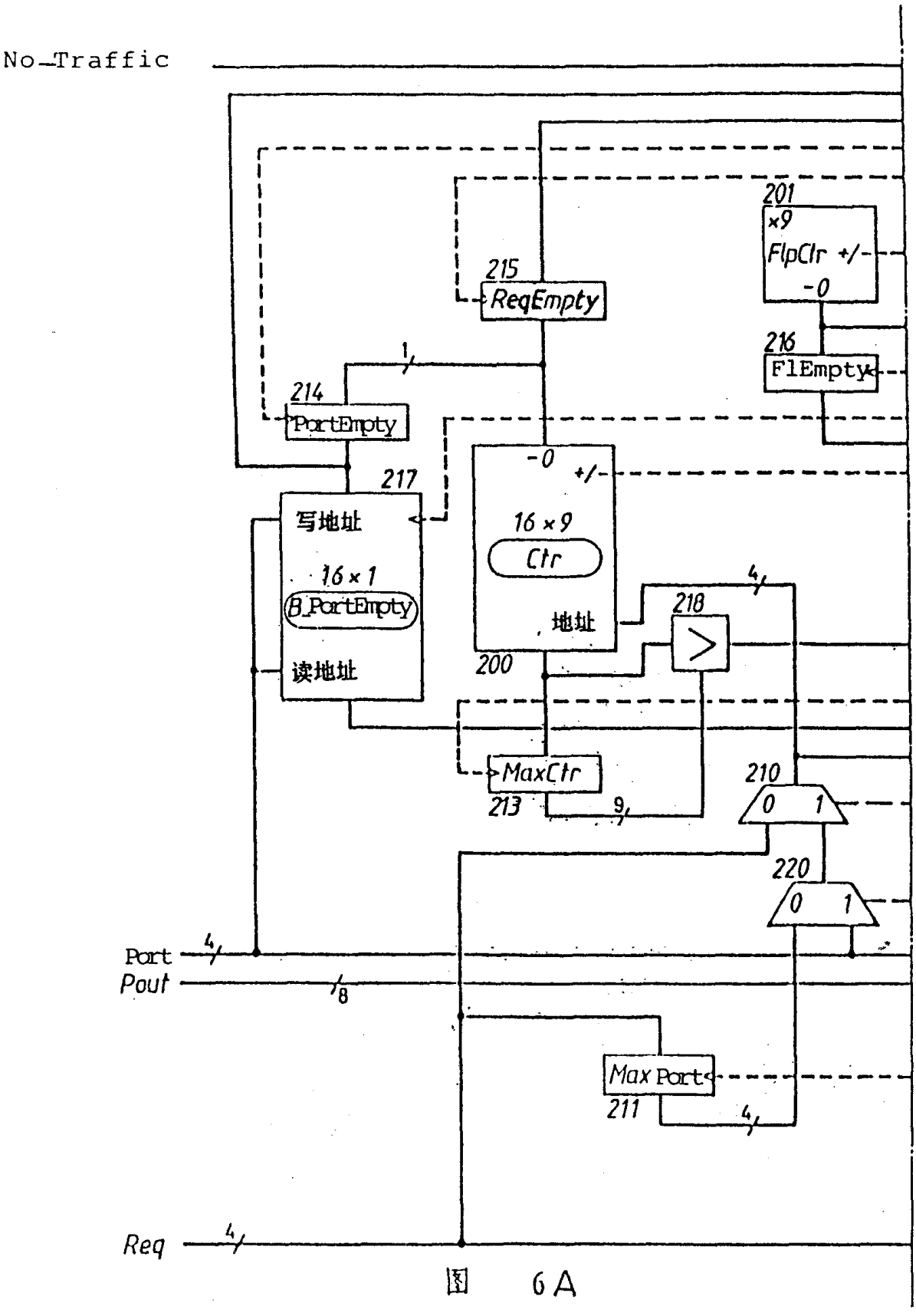

No_Traffic
201
×9
FlpClr +/-
-0
215
ReqEmpty
216
FlEmpty
214
PortEmpty
217
写地址
16×1
B_PortEmpty
读地址
-0
+/-
16×9
Ctr
地址
200
218
MaxCtr
213
9
210
0
1
220
0
1
Port
4
Pout
8
Max Port
211
4
Req
4

图 6A

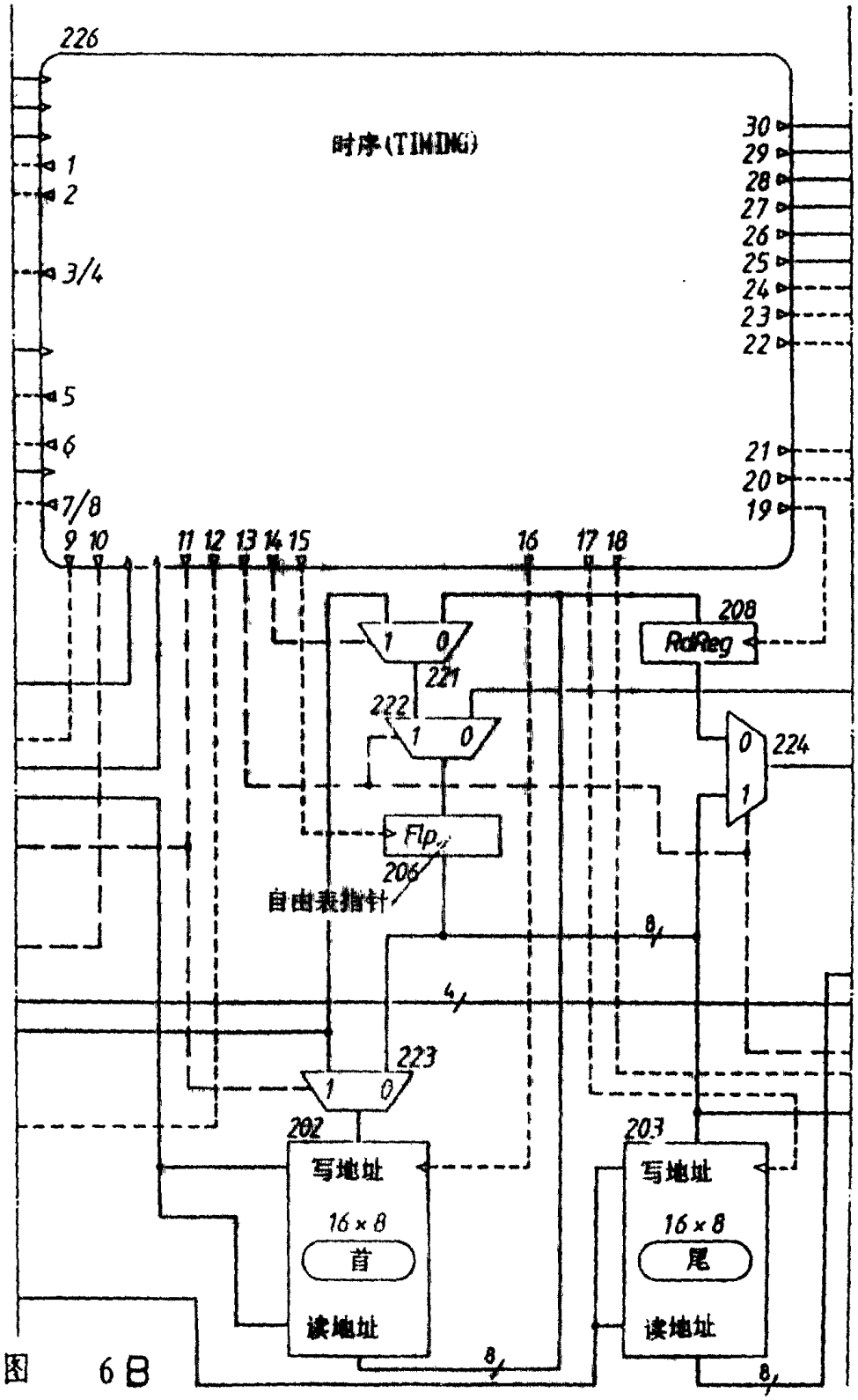

226
时序(TIMING)
1
2
3/4
5
6
7/8
9 10
11 12 13 14 15
16
17 18
30
29
28
27
26
25
24
23
22
21
20
19
208
RdReg
1 0
221
222
1 0
0
1
224
Flp
206
自由表指针
8
4
223
1 0
202
写地址
16 × 8
首
读地址
8
203
写地址
16 × 8
尾
读地址
8

图 6B

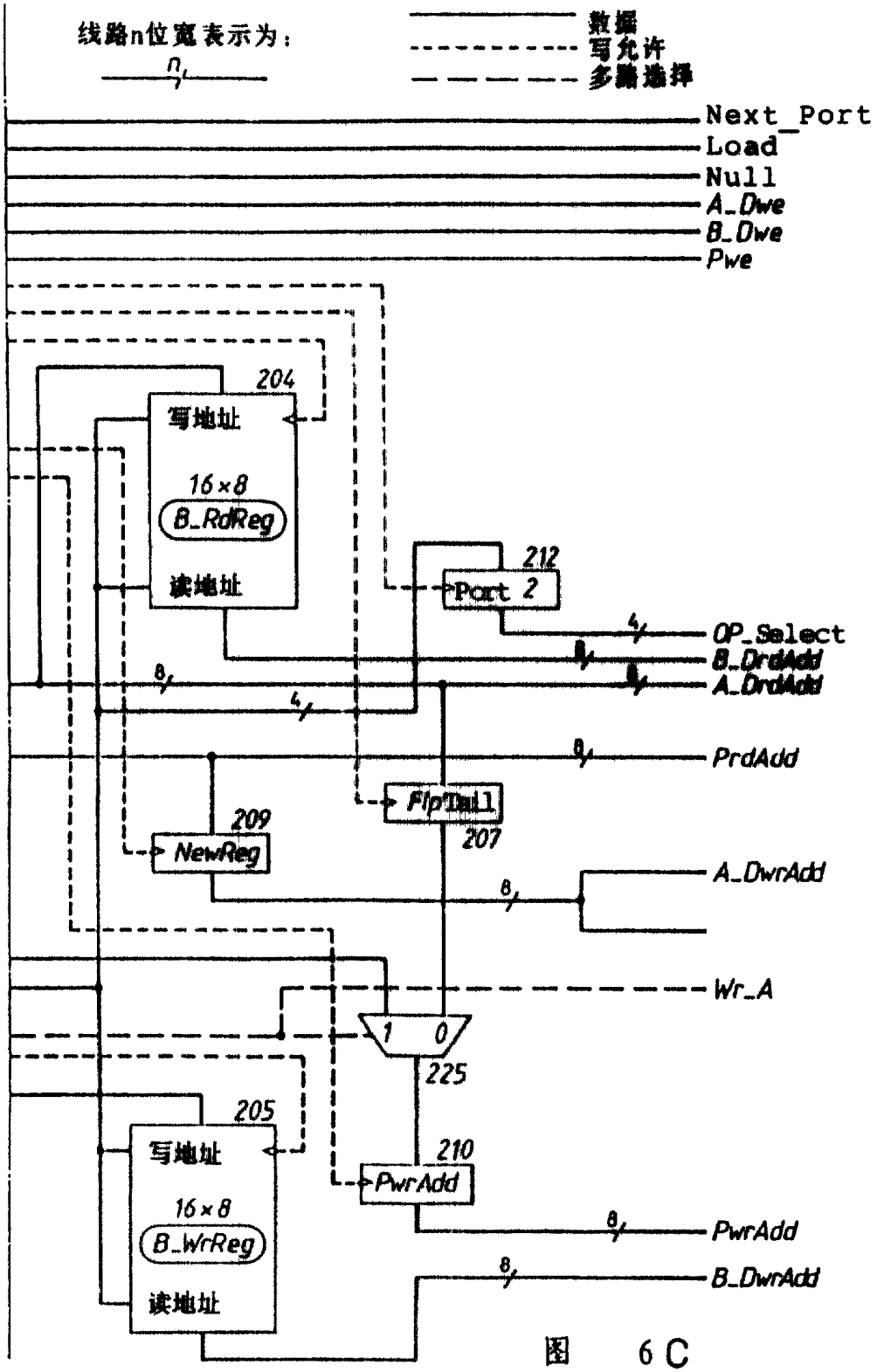

线路n位宽表示为：
n
数据
写允许
多路选择
Next_Port
Load
Null
A_Dwe
B_Dwe
Pwe
204
写地址
16×8
B_RdReg
读地址
212
Port 2
OP_Select
B_DrdAdd
A_DrdAdd
8
4
PrdAdd
FlpTail
207
209
NewReg
A_DwrAdd
Wr_A
1
0
225
205
写地址
16×8
B_WrReg
读地址
210
PwrAdd
PwrAdd
B_DwrAdd

图 6C