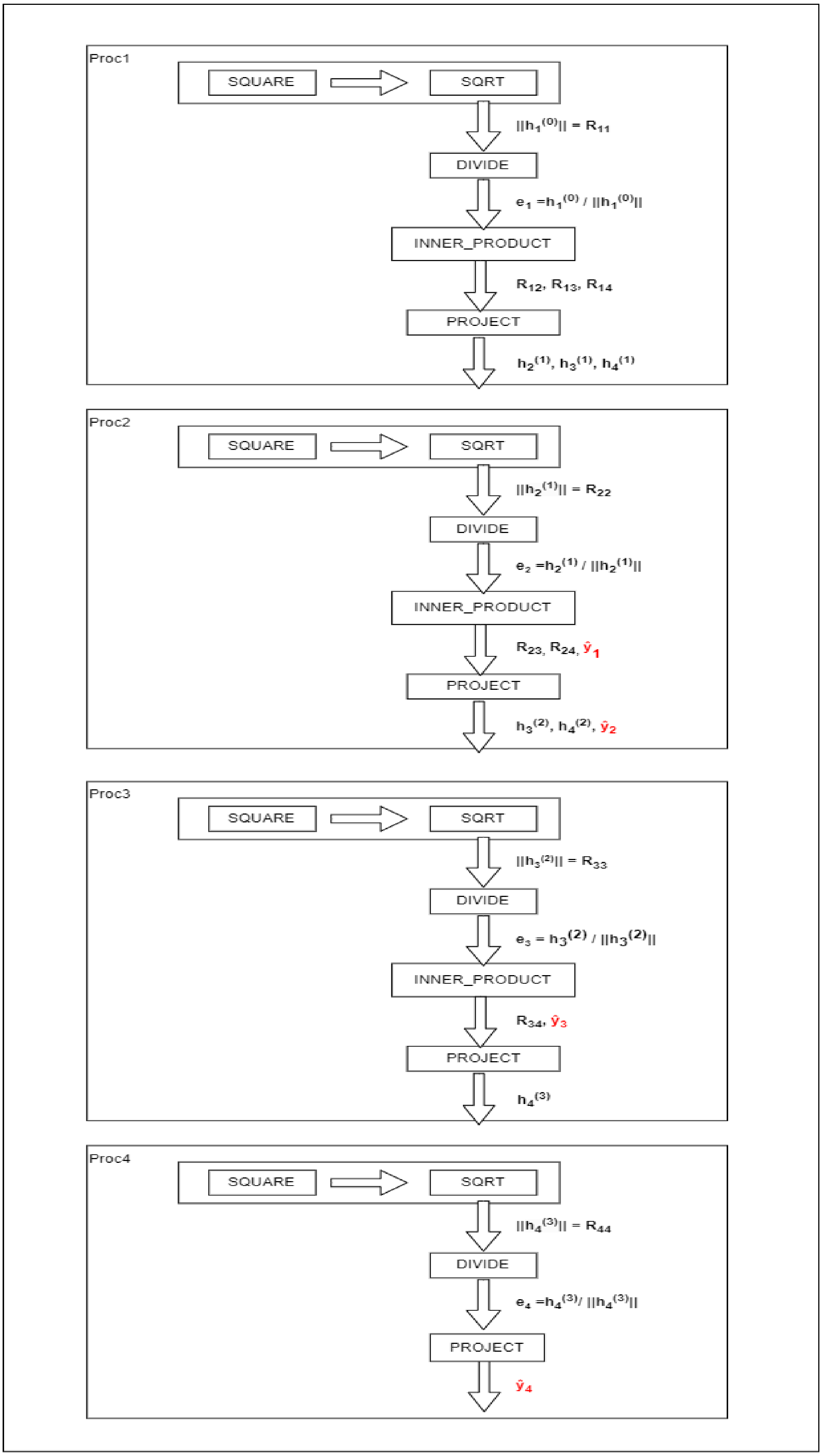
team02_report

- Algorithm

  Our algorithm is mainly based on the concept of modified Gram-Schmidt and some adjustments have been made. For each of resource elements, we divide the algorithm into four processes. And in each process, we separate several computations accordingly. As the picture of our algorithm flow shown below, we have PROC1, PROC2, PROC3, PROC4, and in each process, we have SQUARE, SQRT, DIVIDE, INNER_PRODUCT (not in the PROC4), PROJECTION.

  First, in SQUARE and SQRT, the norm of $\mathbf{h}_1^{(0)}$, $\mathbf{h}_2^{(1)}$, $\mathbf{h}_3^{(2)}$, $\mathbf{h}_4^{(3)}$, namely $\mathbf{R}_{11}$, $\mathbf{R}_{22}$, $\mathbf{R}_{33}$, $\mathbf{R}_{44}$, is computed for each process, respectively. Next, in DIVIDE, $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, $\mathbf{e}_4$ is computed for each process, respectively. Then, in INNER_PRODUCT, $\mathbf{R}_{12}$, $\mathbf{R}_{13}$, $\mathbf{R}_{14}$ is computed for PROC1, $\mathbf{R}_{23}$, $\mathbf{R}_{24}$, $\hat{\mathbf{y}}_1$ for PROC2, and $\mathbf{R}_{34}$, $\hat{\mathbf{y}}_3$ for PROC3. At last, in PROJECTION, $\mathbf{h}_2^{(1)}$, $\mathbf{h}_3^{(1)}$, $\mathbf{h}_4^{(1)}$ is computed for PROC1, $\mathbf{h}_3^{(2)}$, $\mathbf{h}_4^{(3)}$, $\hat{\mathbf{y}}_2$ for PROC2, $\mathbf{h}_4^{(3)}$ for PROC3, and $\hat{\mathbf{y}}_3$ for PROC4.

  The adjustment we make to modified Gram-Schmidt is that we separate the computation of 4 elements of $\hat{\mathbf{y}}$ into different stages and compute them earlier.

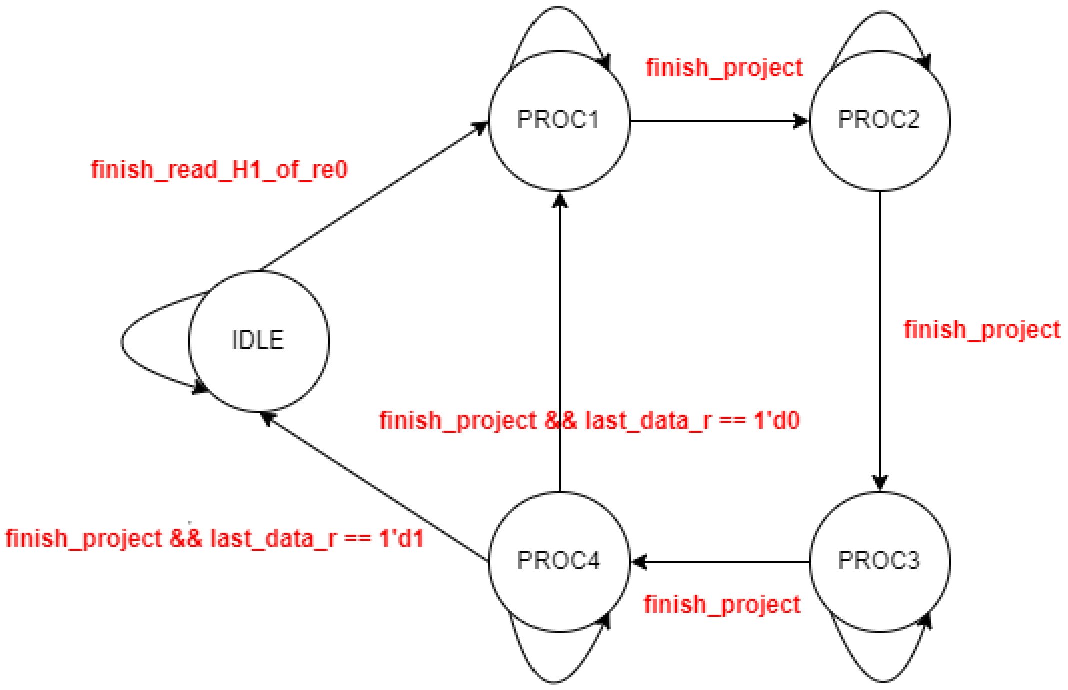


Proc1
SQUARE
SQRT
$||h_1^{(0)}|| = R_{11}$
DIVIDE
$e_1 = h_1^{(0)} / ||h_1^{(0)}||$
INNER_PRODUCT
R_{12}, R_{13}, R_{14}
PROJECT
$h_2^{(1)}, h_3^{(1)}, h_4^{(1)}$
Proc2
SQUARE
SQRT
$||h_2^{(1)}|| = R_{22}$
DIVIDE
$e_2 = h_2^{(1)} / ||h_2^{(1)}||$
INNER_PRODUCT
$R_{23}, R_{24}, \hat{y}_1$
PROJECT
$h_3^{(2)}, h_4^{(2)}, \hat{y}_2$
Proc3
SQUARE
SQRT
$||h_3^{(2)}|| = R_{33}$
DIVIDE
$e_3 = h_3^{(2)} / ||h_3^{(2)}||$
INNER_PRODUCT
$R_{34}, \hat{y}_3$
PROJECT
$h_4^{(3)}$
Proc4
SQUARE
SQRT
$||h_4^{(3)}|| = R_{44}$
DIVIDE
$e_4 = h_4^{(3)} / ||h_4^{(3)}||$
PROJECT
$\hat{y}_4$

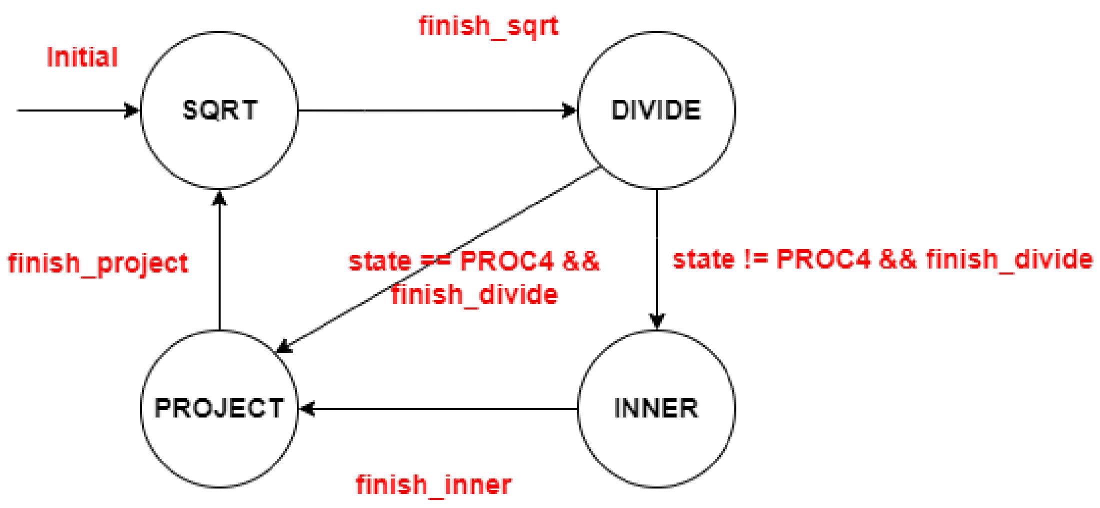
- Hareware implementation
    - Hardware scheduling

    FSM:

    We use four states, PROC1, PROC2, PROC3, PROC4, to represent iteration 0, 1, 2, 3 in modified Gram-Schmidt. And, state IDLE is for initial state.



Then, in every process (PROC1, PROC2, PROC3, PROC4), we need four substates (SQRT, DIVIDE, INNER, PROJECT) to perform operations such as square root, divide, inner product, and projection

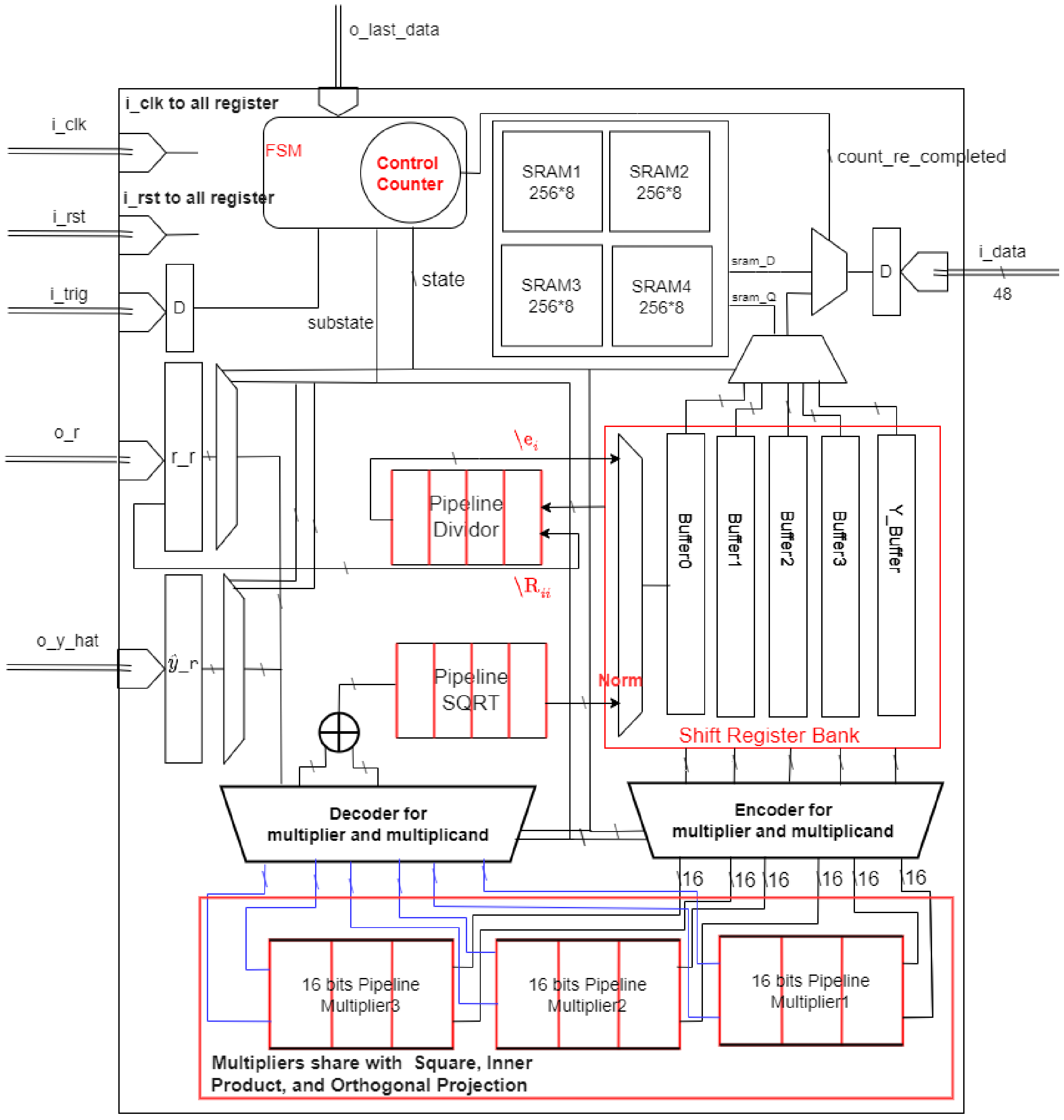


Control Counter:

Because we need at most 20 cycles for each substate in pipeline scenario, we use 5-bit Control Counter to count the cycles we have gone through, precisely controlling how many cycles we should stall.

The control signals, such as finish_sqrt, finish_divide, finish_inner, finish_project, are controlled by the control counter.

- Hardware block diagram



- Area / Power / Latency report

1. Area

    a. At the beginning, we used 6 SRAMs to store data, with each 48-bit data stored in a separate one. Later, we decided to sacrifice the precision of the real and imaginary parts of each data, each with 8 bits of fraction loss. We reduced the number of SRAMs used for each data to 4, saving a total of 2 SRAMs, approximately 40000 µm^2 in area.

b. By discarding 16 bits of each 48-bit data, the size of the data buffer was also reduced. For a resource element consisting of 20 data, the buffer size decreased from 48*20 = 960 bits to 32*20 = 640 bits, resulting in a smaller area occupied by flip-flops.
c. When calculating SQUARE, INNER PRODUCT, and PROJECT, we employed hardware sharing techniques by using DesignWare's multiplier. This reduced the number of multipliers to 3 and shared the adders and subtractors used in the computation process.
d. Without compromising too much on precision, we reduced the number of bits used in various calculations. This reduction aimed to minimize the number of registers and computational logic used during operations, storage of data, and when employing DesignWare pipelined macros, thereby decreasing the number of flip-flops and the area of logical operations.
e. In sequentially feeding buffer data into various processing units such as multipliers or dividers, or when storing data into the buffer, we considered using a shift register instead of an encoder or decoder to avoid excessive area consumption.

f. To reduce the number of comparators and overall area, we consolidated identical if-else control conditions under the same control signal.
g. During the APR (automatic place and route) process, we increased the floorplan's core utilization from 0.7 to 0.9, resulting in a smaller final core area.

2. Power
   a. Due to the computation in each iteration being divided into stages such as square & sqrt, divide, multiply, etc., with only one operation taking place at a time, we set the inputs of idle computation units to 0, reducing switching power.
   b. For each unused DesignWare Pipelined macro, we set its enable signal to low, causing its internal flip-flops to cease operation, thereby saving power in registers.
   c. By storing only 32 bits of a single data entry, the number of used buffers and flip-flops decreased, leading to a reduction in register power.
   d. After sacrificing precision, both the logic circuits during computation and the buffers used in intermediate processes became smaller, resulting in a decrease in both switching power and register power.
   e. We observed that keeping the SRAM's CEN signal constantly active would result in an additional power consumption of around 10mW. Therefore, we controlled

the CEN signal to activate only when necessary, saving additional power caused by SRAM.

f. Utilizing clock gating techniques, we implemented control signals to update various buffers and shift registers only when needed, saving unnecessary power consumption.

3. Latency
   a. We utilize the pipelined version of the DesignWare macro and configure the pipelining stages to 5, allowing the clock frequency to be increased to over 200MHz.
   b. By sacrificing the precision of data during the computation process, we shorten the path of each combinational logic, thereby reducing the clock cycle time and latency.
   c. We buffer the input data with registers before feeding it in, reducing the impact of input latency and consequently lowering the clock cycle time and latency.
   d. Upon output from various computations, we buffer the results with registers to reduce the impact of output latency or shorten timing paths, thus reducing both clock cycle time and latency.