Project Presentation I: Accelerating Bitap on FPGA

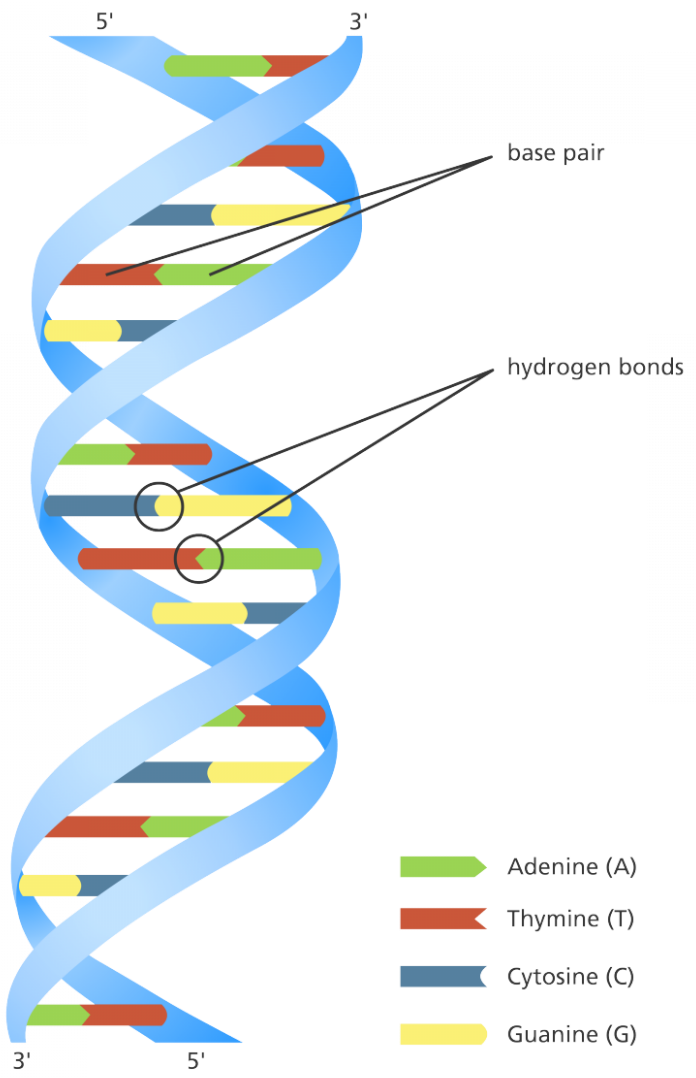
Gayatri Madduri
Sahithi Reddy

Genome Sequence Analysis

- Genome sequence analysis refers to the process of subjecting a DNA or RNA sequence to any of a wide range of analytical methods to understand its features, function, structure, or evolution.

- The first step in genome sequence analysis is the extraction of a genomic sequence such as DNA.

- A DNA consists of two complementary chains, each of which is a very long sequence of four bases: Adenine(A), Thymine(T), Guanine(G),Cytosine(C). T is complementary to A while C is complementary to G. These complementary chains form a double helix structure. Each chain is typically billions of base pairs long.



Extracting DNA: First step in the analysis

- Existing technologies cannot extract the very long DNA sequences as a single unit. Instead, they work by fragmenting the whole sequence into very small overlapping fragments called reads.
- The sequence of bases in these reads is determined using a variety of scientific methods. The length of these reads vary from 20 to 50 base pairs to 150 base pairs.
- This means a huge number of reads need to be assembled back to get the whole genome, which can be used for further analysis.
- The overlapping reads can be advantageous to assemble them back. This method is called De Novo assembly.
- It is also possible to assemble with the help of aa reference genome. This involves looking for possible locations where the read can be aligned in the reference genome.

Approximate string matching(ASM)

- <u>Read mapping</u>:

Takes each read, aligns it to one or more possible locations within the reference genome, and finds the matches and differences (i.e., distance) between the read and the reference genome segment at that location.

Due to genetic variations and sequencing errors, when we look for matching the reference genome and the read, we cannot look for an exact match. Hence, we need to do ASM when we are looking for potential locations to map the read in the reference genome.

- <u>Whole genome analysis:</u>

When we already have two or more full genomes, we want to compute the edit distance to see how similar/dissimilar these genomes are.

Read Mapping

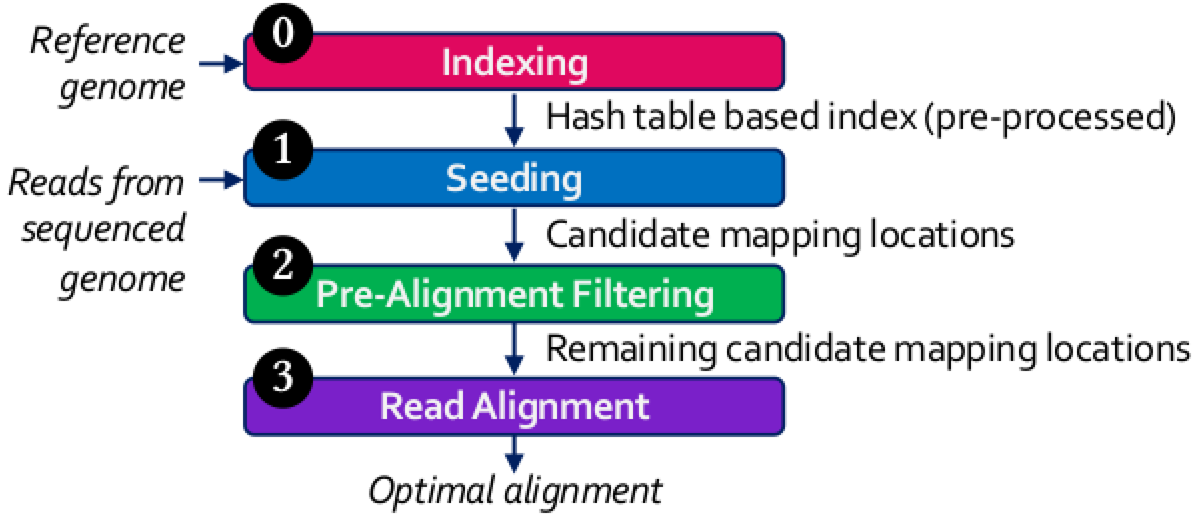


Figure 1. Four steps of read mapping.

Bitap Algorithm

- Bitap tackles the problem of computing the minimum edit distance between a reference text (e.g., reference genome) and a query pattern (e.g., read) with a maximum of k many errors.
- What does it return? It returns the possible alignments such that the edit distance is within k.
- Edit distance can be either deletion, substitution or insertion at a given position of the query genome.

Algorithm 1 Bitap Algorithm

Inputs: `text` (reference), `pattern` (query), `k` (edit distance threshold)
Outputs: `startLoc` (matching location), `editDist` (minimum edit distance)

```
n ← length of reference text
m ← length of query pattern
procedure PRE-PROCESSING
    PM ←generatePatternBitmaskACGT(pattern)      ▷ pre-process the pattern
    for d in 0:k do
        R[d] ← 111..111                          ▷ initialize R bitvectors to 1s
procedure EDIT DISTANCE CALCULATION
    for i in (n-1):-1:0 do                       ▷ iterate over each text character
        curChar ← text[i]
        for d in 0:k do
            oldR[d] ← R[d]                       ▷ copy previous iterations' bitvectors as oldR
        curPM ← PM[curChar]                      ▷ retrieve the pattern bitmask
        R[0] ← (oldR[0]<<1) | curPM              ▷ status bitvector for exact match
        for d in 1:k do                          ▷ iterate over each edit distance
            deletion (D) ← oldR[d-1]
            substitution (S) ← (oldR[d-1]<<1)
            insertion (I) ← (R[d-1]<<1)
            match (M) ← (oldR[d]<<1) | curPM
            R[d] ← D & S & I & M                 ▷ status bitvector for d errors
        if MSB of R[d] == 0, where 0 ≤ d ≤ k     ▷ check if MSB is 0
            startLoc ← i                         ▷ matching location
            editDist ← d                         ▷ found minimum edit distance
```

Step 1: PATTERN MASK

- Text: **CGTGA**
- Query: **CTGA**
- Edit distance threshold=1

	C	**T**	**G**	**A**
PM(A)	1	1	1	0
PM(G)	1	1	0	1
PM(C)	0	1	1	1
PM(T)	1	0	1	1

STEP 2:STATUS BITVECTORS

- R --- a (k+1) x m matrix
- K=edit distance threshold
- M=query length
- R[d][j] represents holds the partial match information between text[i : (n–1)] and the query with maximum of d errors. Value of R[d][j] can either be 0 or 1.

1	1	1	1
1	1	1	1

STEP 3: EXACT MATCHING

- Represented by R[0](first row of R)
- We perform this operation for every I from the end of reference to beginning of reference.
- For a given I, R[0][j]=0 if read[j..m-1]=reference[I…n-1]
- (R[0]=OldR[0]<<1) | PM(reference[I])

K=0 ref: CGTGACT Read: TGAC

k=0 exact matching

Ex 1

Reference:- CGTGACT   Read:- TGAC

Pm(A) = 1101

Pm(C) = 1110

Pm(G) = 1011

Pm(T) = 0111

Iteration	
1	CGTGACT TGAC 1110 | 0111 = 1111 → nomatch
2	CGTGACT TGAC 1110 | 1110 = 1110 → match
3	CGTGACT TGAC 1100 | 1101 = 1101 → match
4	CGTGACT TGAC 1010 | 1011 = 1011 → match
5	CGTGACT 10110 | 0111 = 0111 → match

K=0
ref: cgtgac
read: ggac

k=0 exact matching

CGTGAC GGAC

pm(A) = 1101
pm(T) = 1111
pm(C) = 1110
pm(G) = 0011

Example 2

Iterations

CGTGAC | GGAC

1. 1110 | 1110 = 1110 → match

2. CGTGAC | GGAC
1100 | 1101 = 1101 → match

3. CGTGAC | GGAC
1010 | 0011 = 1011 → match

4. CGTGAC | GGAC
~~1010 | 0011 = 1011~~
0110 | 1111 = 1111 → mismatch

5. CGTGAC | GGAC
1110 | 0011 = 1111 → mismatch

STEP 4: Calculating r[d]

- R[d] is similar to R[0] but here we give space for deletion/insertion/substitution and exact match.
- Hence, we calculate all the four possibilites.
- If any of the deletions/insertions/substitutions cause an exact match it must be reflected in R[d]. Hence, we and all the results
- R[d]=D&S&I&M

$k > 0$

MATCH CASE :-

$oldR[d] << 1 \mid pm[char]$

↓

has information from i to n-1

⇒ if its a match then i-1 to n-1.

SUBSTITUTION :-

$oldR[d-1] << 1$

→ consume one character from reference as that is what you will be substituting with.

INSERTION :-

$R[d-1] << 1$

→ whatever is currently mismatched, just move it assuming its a match.

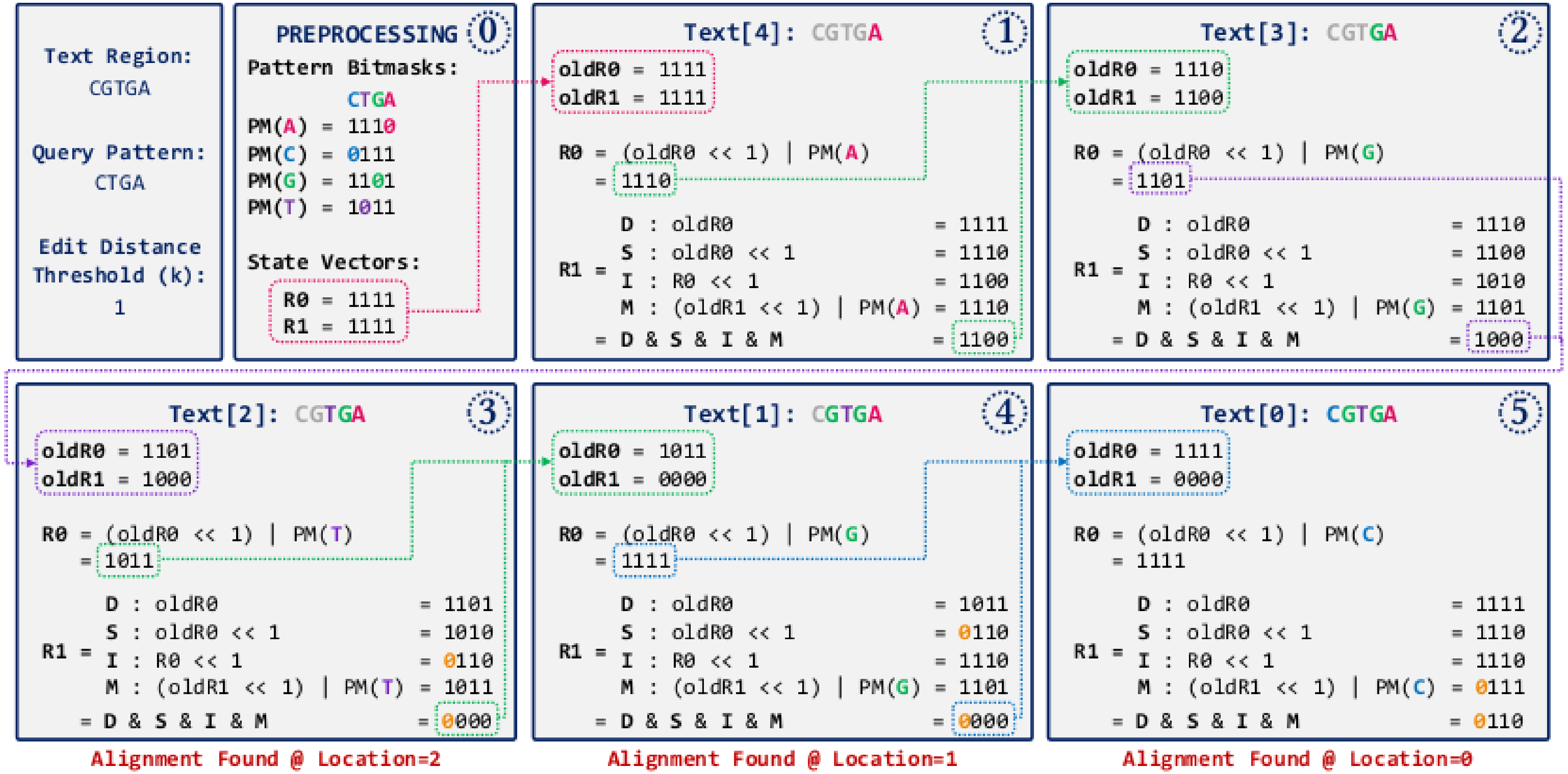
DELETION :-

$oldR[d-1]$

STEP 5:STORING PATTERN MATCHES

- If leftmost bit of R[d] becomes 0 for some 0<=d<=k for some position I in the reference genome, we can say that the read matches with the reference at position I with at most d errors.
- Hence, when this condition is obtained, we store I and d.

Example

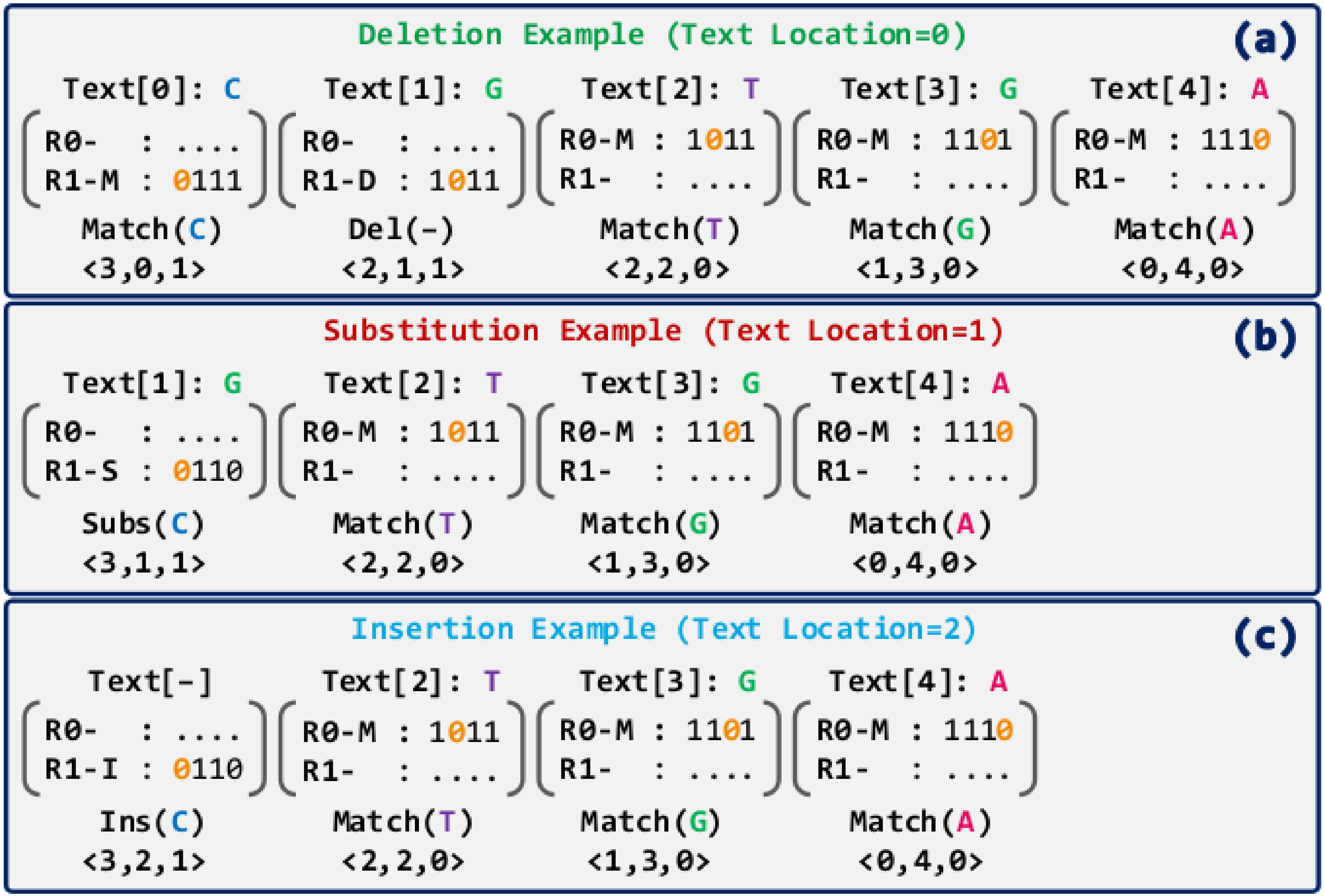

Text Region:
CGTGA
Query Pattern:
CTGA
Edit Distance Threshold (k):
1
PREPROCESSING
0
Pattern Bitmasks:
CTGA
PM(A) = 1110
PM(C) = 0111
PM(G) = 1101
PM(T) = 1011
State Vectors:
R0 = 1111
R1 = 1111
Text[4]: CGTGA
1
oldR0 = 1111
oldR1 = 1111
R0 = (oldR0 << 1) | PM(A)
= 1110
D : oldR0 = 1111
S : oldR0 << 1 = 1110
I : R0 << 1 = 1100
M : (oldR1 << 1) | PM(A) = 1110
R1 =
= D & S & I & M = 1100
Text[3]: CGTGA
2
oldR0 = 1110
oldR1 = 1100
R0 = (oldR0 << 1) | PM(G)
= 1101
D : oldR0 = 1110
S : oldR0 << 1 = 1100
I : R0 << 1 = 1010
M : (oldR1 << 1) | PM(G) = 1101
R1 =
= D & S & I & M = 1000
Text[2]: CGTGA
3
oldR0 = 1101
oldR1 = 1000
R0 = (oldR0 << 1) | PM(T)
= 1011
D : oldR0 = 1101
S : oldR0 << 1 = 1010
I : R0 << 1 = 0110
M : (oldR1 << 1) | PM(T) = 1011
R1 =
= D & S & I & M = 0000
Text[1]: CGTGA
4
oldR0 = 1011
oldR1 = 0000
R0 = (oldR0 << 1) | PM(G)
= 1111
D : oldR0 = 1011
S : oldR0 << 1 = 0110
I : R0 << 1 = 1110
M : (oldR1 << 1) | PM(G) = 1101
R1 =
= D & S & I & M = 0000
Text[0]: CGTGA
5
oldR0 = 1111
oldR1 = 0000
R0 = (oldR0 << 1) | PM(C)
= 1111
D : oldR0 = 1111
S : oldR0 << 1 = 1110
I : R0 << 1 = 1110
M : (oldR1 << 1) | PM(C) = 0111
R1 =
= D & S & I & M = 0110
Alignment Found @ Location=2
Alignment Found @ Location=1
Alignment Found @ Location=0

- Ref: C G T G A
- Read: C - T G A   edit distance : 1

- Ref: C G T G A
- Read: _ C T G A   edit distance : 2

- Ref: C G _ T G A
- Read: _ _ C T G A   edit distance : 3

Figure 2. Three types of errors (i.e., edits).



Deletion Example (Text Location=0)
(a)
Text[0]: C
R0- : ....
R1-M : 0111
Match(C)
<3,0,1>
Text[1]: G
R0- : ....
R1-D : 1011
Del(-)
<2,1,1>
Text[2]: T
R0-M : 1011
R1- : ....
Match(T)
<2,2,0>
Text[3]: G
R0-M : 1101
R1- : ....
Match(G)
<1,3,0>
Text[4]: A
R0-M : 1110
R1- : ....
Match(A)
<0,4,0>
Substitution Example (Text Location=1)
(b)
Text[1]: G
R0- : ....
R1-S : 0110
Subs(C)
<3,1,1>
Text[2]: T
R0-M : 1011
R1- : ....
Match(T)
<2,2,0>
Text[3]: G
R0-M : 1101
R1- : ....
Match(G)
<1,3,0>
Text[4]: A
R0-M : 1110
R1- : ....
Match(A)
<0,4,0>
Insertion Example (Text Location=2)
(c)
Text[-]
R0- : ....
R1-I : 0110
Ins(C)
<3,2,1>
Text[2]: T
R0-M : 1011
R1- : ....
Match(T)
<2,2,0>
Text[3]: G
R0-M : 1101
R1- : ....
Match(G)
<1,3,0>
Text[4]: A
R0-M : 1110
R1- : ....
Match(A)
<0,4,0>

LOOP DEPENDENCY REMOVAL

```
Text Region:
CGTGA

Query Pattern:
CTGA

Edit Distance
Threshold (k):
1
```

```
PREPROCESSING                                  0
Pattern Bitmasks:
        CTGA
PM(A) = 1110
PM(C) = 0111
PM(G) = 1101
PM(T) = 1011

State Vectors:
  R0 = 1111
  R1 = 1111
```

```
Text[4]: CGTGA                                 1
oldR0 = 1111
oldR1 = 1111

R0 = (oldR0 << 1) | PM(A)
   = 1110

       D : oldR0                    = 1111
R1 =   S : oldR0 << 1               = 1110
       I : R0 << 1                  = 1100
       M : (oldR1 << 1) | PM(A)     = 1110
   = D & S & I & M                  = 1100
```

```
Text[3]: CGTGA                                 2
oldR0 = 1110
oldR1 = 1100

R0 = (oldR0 << 1) | PM(G)
   = 1101

       D : oldR0                    = 1110
R1 =   S : oldR0 << 1               = 1100
       I : R0 << 1                  = 1010
       M : (oldR1 << 1) | PM(G)     = 1101
   = D & S & I & M                  = 1000
```

```
Text[2]: CGTGA                                 3
oldR0 = 1101
oldR1 = 1000

R0 = (oldR0 << 1) | PM(T)
   = 1011

       D : oldR0                    = 1101
R1 =   S : oldR0 << 1               = 1010
       I : R0 << 1                  = 0110
       M : (oldR1 << 1) | PM(T)     = 1011
   = D & S & I & M                  = 0000
```

Alignment Found @ Location=2

```
Text[1]: CGTGA                                 4
oldR0 = 1011
oldR1 = 0000

R0 = (oldR0 << 1) | PM(G)
   = 1111

       D : oldR0                    = 1011
R1 =   S : oldR0 << 1               = 0110
       I : R0 << 1                  = 1110
       M : (oldR1 << 1) | PM(G)     = 1101
   = D & S & I & M                  = 0000
```

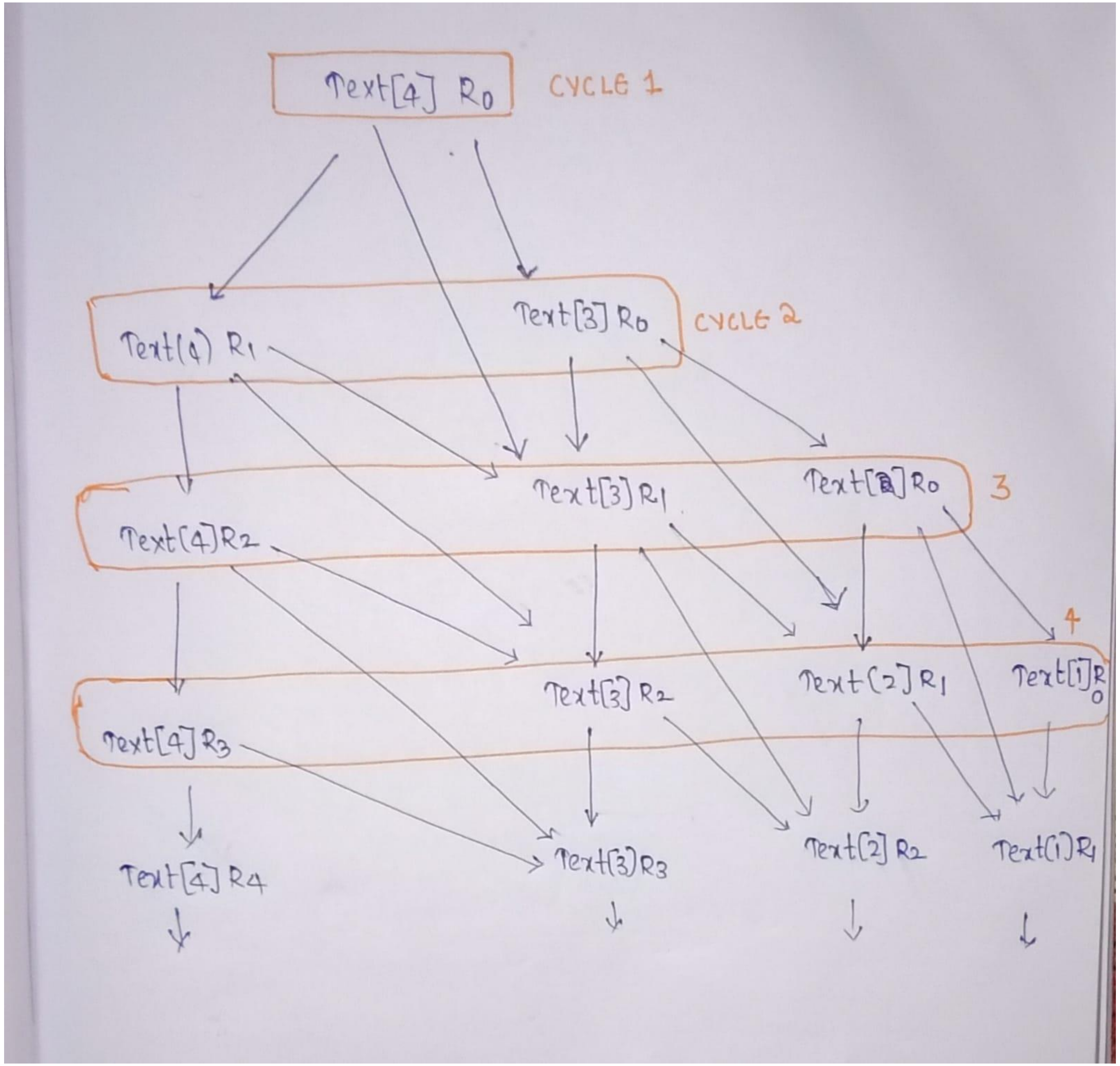
Alignment Found @ Location=1

```
Text[0]: CGTGA                                 5
oldR0 = 1111
oldR1 = 0000

R0 = (oldR0 << 1) | PM(C)
   = 1111

       D : oldR0                    = 1111
R1 =   S : oldR0 << 1               = 1110
       I : R0 << 1                  = 1110
       M : (oldR1 << 1) | PM(C)     = 0111
   = D & S & I & M                  = 0110
```

Alignment Found @ Location=0



Cycle#	Thread1 *R0/1/2/..*
#1	T0-R0
...	...
#8	T0-R7
#9	T1-R0
...	...
#16	T1-R7
#17	T2-R0
...	...
#24	T2-R7
#25	T3-R0
...	...
#32	T3-R7

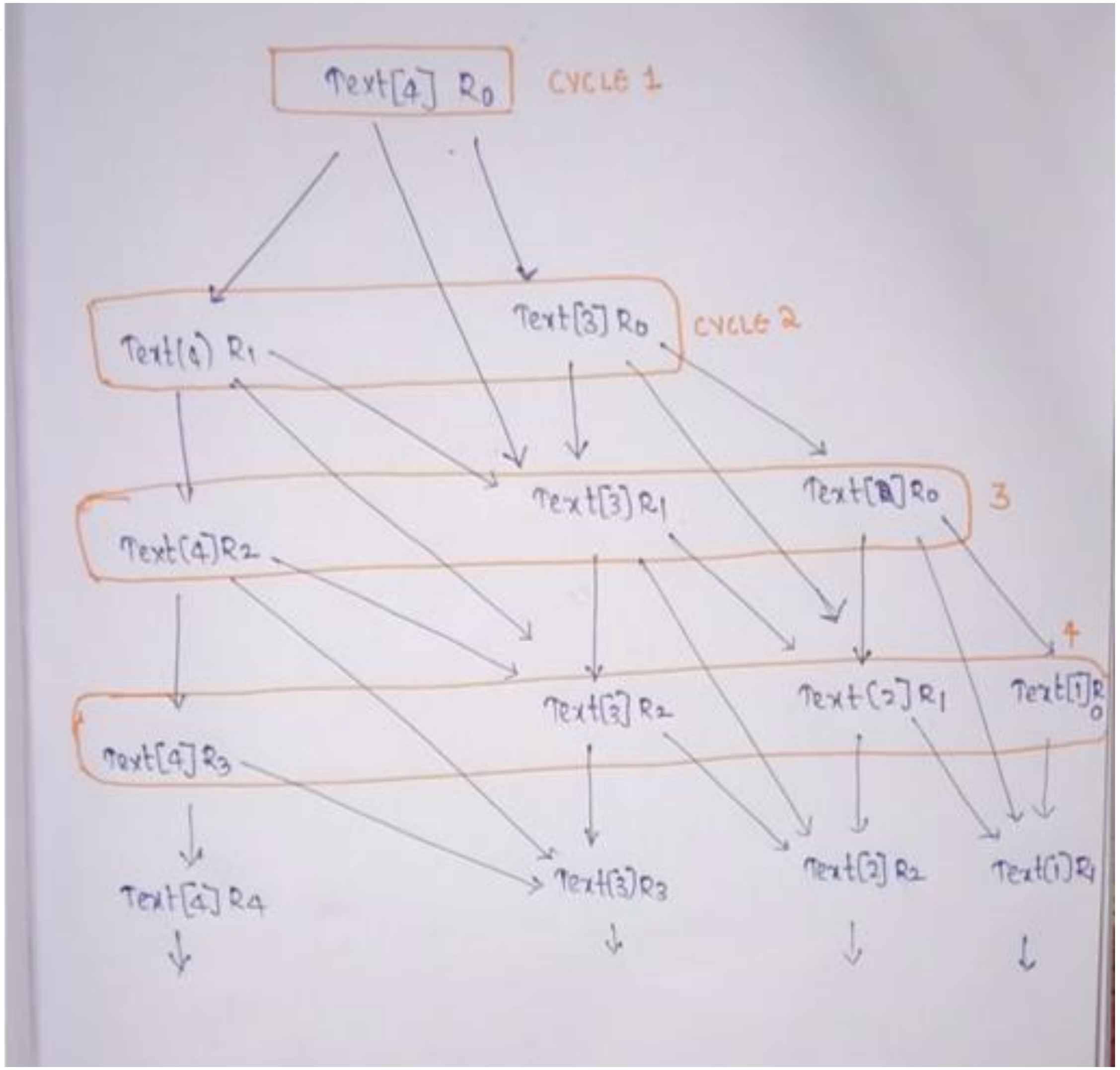
Cycle#	Thread1 *R0/4*	Thread2 *R1/5*	Thread3 *R2/6*	Thread4 *R3/7*
#1	T0-R0	–	–	–
#2	T1-R0	T0-R1	–	–
#3	T2-R0	T1-R1	T0-R2	–
#4	T3-R0	T2-R1	T1-R2	T0-R3
#5	T0-R4	T3-R1	T2-R2	T1-R3
#6	T1-R4	T0-R5	T3-R2	T2-R3
#7	T2-R4	T1-R5	T0-R6	T3-R3
#8	T3-R4	T2-R5	T1-R6	T0-R7
#9	–	T3-R5	T2-R6	T1-R7
#10	–	–	T3-R6	T2-R7
#11	–	–	–	T3-R7

data *written to memory*
data *read from memory*
target cell (R_d)
cells target cell depends on ($oldR_d$, R_{d-1}, $oldR_{d-1}$)

Algorithm 1 Bitap Algorithm

Inputs: text (reference), pattern (query), k (edit distance threshold)
Outputs: startLoc (matching location), editDist (minimum edit distance)

```
n ← length of reference text
m ← length of query pattern
procedure PRE-PROCESSING
    PM ←generatePatternBitmaskACGT(pattern)        ▷ pre-process the pattern
    for d in 0:k do
        R[d] ← 111..111                            ▷ initialize R bitvectors to 1s
procedure EDIT DISTANCE CALCULATION
    for i in (n-1):-1:0 do                         ▷ iterate over each text character
        curChar ← text[i]
        for d in 0:k do
            oldR[d] ← R[d]                         ▷ copy previous iterations' bitvectors as oldR
        curPM ← PM[curChar]                        ▷ retrieve the pattern bitmask
        R[0] ← (oldR[0]<<1) | curPM                ▷ status bitvector for exact match
        for d in 1:k do                            ▷ iterate over each edit distance
            deletion (D) ← oldR[d-1]
            substitution (S) ← (oldR[d-1]<<1)
            insertion (I) ← (R[d-1]<<1)
            match (M) ← (oldR[d]<<1) | curPM
            R[d] ← D & S & I & M                   ▷ status bitvector for d errors
        if MSB of R[d] == 0, where 0 ≤ d ≤ k       ▷ check if MSB is 0
            startLoc ← i                           ▷ matching location
            editDist ← d                           ▷ found minimum edit distance
```



Cycle : units

- 1 st : (n)(0)
- 2nd: (n)(1) + (n-1)(0)
- K+1 th iteration: (n)(k),(n-1)(k-1),(n-2)(k-2)….,(n-k)(0)
- K+2 th : (n-1)k ,(n-2)(k-1)…………..(n-k+1)(0)
- K+3 : (n-2)(k),(n-3)(k-1),………….(n-k+2)(0)
- K+x th : (n-(x-1))(k),……(n-k+(x-1))(0)
- If K+xth was last (n-(x-1)) == 1 ; x=n
- No.of iterations = K+n

Psuedo code:

```
For j in range(K+n) :
    For I in range(min(j,k),0,-1):
     #loop unroll
      Compute test[n-j+I][I]
       When I!=0
        Do
        I , S , M, D and compute

     #bind jth and (j-1)th values  to use
in
       (J+1)th iteration—do it in parallel
pipelined
```

INTO THE CODE

Burst read

```
extern "C"
{
    void wide_vadd(
        unsigned long long  *pm,
        const char *pattern,
        const char *text,
        int *k,
        const int *m,
        const int *n,
        int *out       // Output Result
    )
    {
#pragma HLS INTERFACE m_axi port = pm max_read_burst_length = 64  offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = pattern max_read_burst_length = 8  offset = slave bundle = gmem1
#pragma HLS INTERFACE m_axi port = text max_write_burst_length = 8 offset = slave bundle = gmem1
#pragma HLS INTERFACE m_axi port = k max_write_burst_length = 64 offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = m max_write_burst_length = 64 offset = slave bundle = gmem2
#pragma HLS INTERFACE m_axi port = n max_write_burst_length = 64 offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = out max_write_burst_length = 64 offset = slave bundle = gmem3
#pragma HLS INTERFACE s_axilite port = pm bundle = control
#pragma HLS INTERFACE s_axilite port = out bundle = control
#pragma HLS INTERFACE s_axilite port = pattern bundle = control
#pragma HLS INTERFACE s_axilite port = text bundle = control
#pragma HLS INTERFACE s_axilite port =k bundle = control
#pragma HLS INTERFACE s_axilite port = m bundle = control
#pragma HLS INTERFACE s_axilite port = n bundle = control
#pragma HLS INTERFACE s_axilite port = return bundle = control

    //create local buffers

   int patt_len;
   int text_len;
   unsigned long long patternbitmasks[8];
   char patt[100];
   char tex[100];
```

Bind to BRAM

Bind ed all the

inputs to BRAM and chose

RAM_2P, which supports read to a port and read & write to another port.

```
    //create local buffers

   int patt_len;
   int text_len;
   unsigned long long patternbitmasks[8];
   char patt[100];
   char tex[100];

#pragma HLS BIND_OP variable =patternbitmasks op=mul impl=DSP latency=2
#pragma HLS BIND_OP variable=patt op=add impl=DSP latency=2
#pragma HLS BIND_OP variable=tex op=add impl=DSP latency=2
#pragma HLS BIND_OP variable=patt_len op=add impl=DSP latency=2
#pragma HLS BIND_OP variable=text_len op=add impl=DSP latency=2

// BINDING TO BRAM
#pragma HLS BIND_STORAGE variable= patternbitmasks type=RAM_2P impl=BRAM latency=2
#pragma HLS BIND_STORAGE variable= patt type=RAM_2P impl=BRAM latency=2
#pragma HLS BIND_STORAGE variable= tex type=RAM_2P impl=BRAM latency=2
#pragma HLS BIND_STORAGE variable=patt_len type=RAM_2P impl=BRAM latency=2
#pragma HLS BIND_STORAGE variable =text_len type=RAM_2P impl=BRAM latency=2
//std:: cout << "---------------------kernel side-------------------"<<"\n";
   patt_len=m[0];
   text_len=n[0];
   int num= ceil(m[0]/(64*1.0));
     num=num*4;
     int val = ceil (patt_len/(64.0));
     //std::cout << val << "is val\n";
  for (int i=0;i<num;i++)
      patternbitmasks[i]=pm[i];

  int offset=k[0];
```

- Bind R's to BRAM
- Sneak peak into Looping

```
#pragma HLS BIND_OP variable=R op=mul impl=DSP latency=2
#pragma  HLS BIND_STORAGE variable=R type=RAM_2P impl=BRAM latency=2
//bind needs specific value hence specified upper bounds.
  if (val==1)
  {
  for (int j=0;j<offset+text_len+1;j++)
      for (int i=min(j-1,offset);i>=0;i--)
      {
#pragma HLS loop unroll
//compute R[n-j+1][i]
          //get current pattern bit mask
          uint64_t currpm;
          int iter = text_len-j+i;
          if (iter<0 or i<0)
              continue;
          else
          {
          if(tex[iter]=='A' or tex[iter]=='a')…
          else if(tex[iter]=='C' or tex[iter]=='c')…
          else if(tex[iter]=='G' or tex[iter]=='g')…
          else if(tex[iter]=='T' or tex[iter]=='t')…

          if(i==0)
          {
              //compute single R with range
              if(iter==text_len-1)
              {   R[iter][0]= oldR[0]<<1;
              //std::cout << "after shift" <<R[iter][0] << "\n";
                  R[iter][0]|=currpm;
              }
              else
              {
                 // std::cout << "before shift"<<R[iter+1][0]<<"\n";
                  R[iter][0]= R[iter+1][0]<<1;
                 // std::cout << "after shift" <<R[iter][0] << "\n";
                  R[iter][0]|=currpm;
```

- **Communication**
  - Map the buffers to kernel space and populate them (same as wide_vadd)
  - Burst read the inputs
  - As extension multiple DDRs can be used.

- **Computations**
  - Parallelism among two strings is exploited
  - Multiple pairs of strings can be processed, but we may get memory overhead.

Results:

```
<terminated> (exit value: 0) SystemDebugger_Bitap_system_1_Bitap [OpenCL] /home/ec2-user/workspace/Bitap/Emulation-HW/Bitap (12/9/21, 6:
TT
CC
GG
CC
AA
TT
CC
GG
offset2CPU result
entered filter
Generating Pattern bit masks
length is8
entered DC
k is2count2
length in DC before R: 6
length in DC : 6
DC return value-10
0
sucess, edit distance is0INFO::[ Vitis-EM 22 ] [Time elapsed: 0 minute(s) 44 seconds, Emulation time: 0.160637 ms
Data transfer between kernel(s) and global memory(s)
wide_vadd_1:m_axi_gmem-DDR[1]          RD = 0.047 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem1-DDR[1]          RD = 0.000 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem2-DDR[1]          RD = 0.004 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem3-DDR[1]          RD = 0.000 KB              WR = 0.004 KB

INFO: [HW-EMU 06-0] Waiting for the simulator process to exit
INFO: [HW-EMU 06-1] All the simulator processes exited successfully
```

vadd_system | vadd_system_... | Console | Executables | Debug Shell | Vitis Log | Problems | Debugger Con... | p_vadd | »21

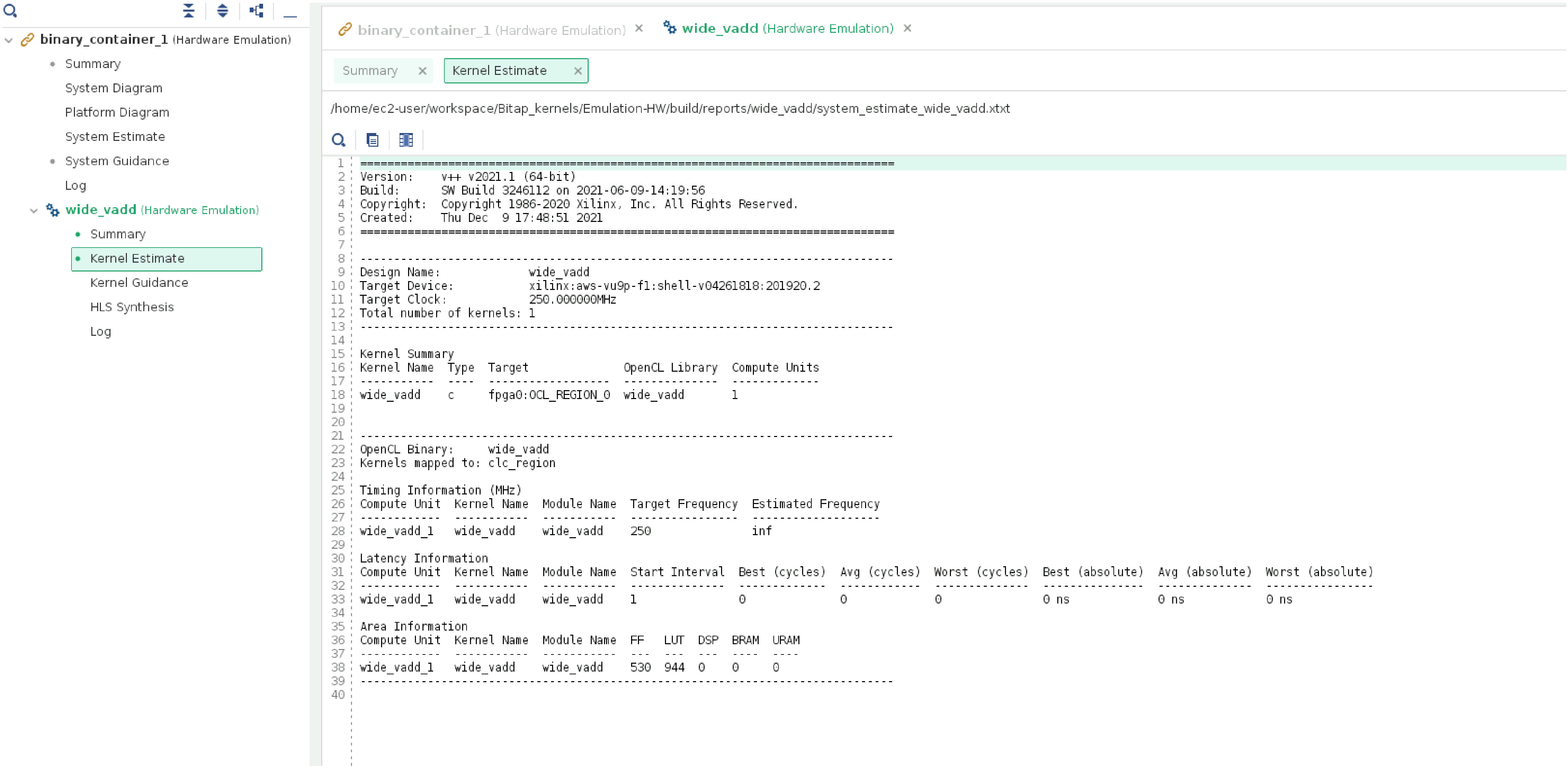
```
<terminated> (exit value: 0) SystemDebugger_Bitap_system_Bitap [OpenCL] /home/ec2-user/workspace/Bitap/Emulation-HW/Bitap (12/9/21, 6:03 PM)
[Console output redirected to file:/home/ec2-user/workspace/Bitap/Emulation-HW/SystemDebugger_Bitap_system_Bitap.launch.log]
-----Accelerating bitap algorithm-----------
Loading/home/ec2-user/workspace/Bitap_system/Emulation-HW/binary_container_1.xclbinto program
INFO: [HW-EMU 01] Hardware emulation runs simulation underneath. Using a large data set will result in long simulation times. It is recommended that a
configuring penguin scheduler mode
scheduler config ert(0), dataflow(1), slots(16), cudma(1), cuisr(0), cdma(0), cus(1)
length of pattern is 4length of text is 7
CPU result
success, edit distance is0INFO::[ Vitis-EM 22 ] [Time elapsed: 0 minute(s) 42 seconds, Emulation time: 0.152899 ms]
Data transfer between kernel(s) and global memory(s)
wide_vadd_1:m_axi_gmem-DDR[1]          RD = 0.016 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem1-DDR[1]          RD = 0.000 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem2-DDR[1]          RD = 0.004 KB              WR = 0.000 KB
wide_vadd_1:m_axi_gmem3-DDR[1]          RD = 0.000 KB              WR = 0.004 KB

INFO: [HW-EMU 06-0] Waiting for the simulator process to exit
INFO: [HW-EMU 06-1] All the simulator processes exited successfully
```

XSCT Console | Emulation Console

QEMU Process

Resources Usage :




```
===============================================================================
Version:    v++ v2021.1 (64-bit)
Build:      SW Build 3246112 on 2021-06-09-14:19:56
Copyright:  Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
Created:    Thu Dec  9 17:48:51 2021
===============================================================================

-------------------------------------------------------------------------------
Design Name:             wide_vadd
Target Device:           xilinx:aws-vu9p-f1:shell-v04261818:201920.2
Target Clock:            250.000000MHz
Total number of kernels: 1
-------------------------------------------------------------------------------

Kernel Summary
Kernel Name  Type  Target              OpenCL Library  Compute Units
-----------  ----  ------------------  --------------  -------------
wide_vadd    c     fpga0:OCL_REGION_0  wide_vadd       1


-------------------------------------------------------------------------------
OpenCL Binary:     wide_vadd
Kernels mapped to: clc_region

Timing Information (MHz)
Compute Unit  Kernel Name  Module Name  Target Frequency  Estimated Frequency
------------  -----------  -----------  ----------------  -------------------
wide_vadd_1   wide_vadd    wide_vadd    250               inf

Latency Information
Compute Unit  Kernel Name  Module Name  Start Interval  Best (cycles)  Avg (cycles)  Worst (cycles)  Best (absolute)  Avg (absolute)  Worst (absolute)
------------  -----------  -----------  --------------  -------------  ------------  --------------  ---------------  --------------  ----------------
wide_vadd_1   wide_vadd    wide_vadd    1               0              0             0               0 ns             0 ns            0 ns

Area Information
Compute Unit  Kernel Name  Module Name  FF   LUT  DSP  BRAM  URAM
------------  -----------  -----------  ---  ---  ---  ----  ----
wide_vadd_1   wide_vadd    wide_vadd    530  944  0    0     0
-------------------------------------------------------------------------------
```

Future work:

- Can be extended to support long reads easily.
- Can be directly read from files
- Multiple DDRs
- Replacing buffers in BRAM

THANK YOU

Gayatri & Sahithi