きわめる。拓く。創り出す。

長崎総合科学大学
Nagasaki Institute of Applied Science

高位合成（HLS)を用いたFPGAによる検出器データ処理のアクセラレーション

田中・大山　研究室所属
電気電子工学コース　4年
1818013　Edwin Tan Yong Yee

2021/9/22　HLS会議　発表

Mean, Median, Mode の計算コード

component

```cpp
#include "HLS/hls.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <numeric>
using namespace std;

component double max(vector<double> data, int n){
    double w = 0;
    double t = w + 1;

    for (int i=0; i<n; i++)
    {
        if (data[i] > w)
        w = data[i];
    }
    return w;
}

component double mean(double sum, int n){
    double x;
    x = sum/n;
    return x;
}

component double median(vector<double> data, int n){
    double y;

    if(n % 2 == 0)
        y = ((data[n/2] + data[n/2 -1]) / 2);
    else
        y = data[n/2];
    return y;
}

component double mode(vector<double> data, int n){

    int max_count = 1, z = data[0], curr_count = 1;
    for (int i = 1; i < n; i++){
        if (data[i] == data[i-1])
            curr_count++;
            else if (curr_count > max_count){
            max_count = curr_count;
            z = data[n-1];
            }

        else{
            if (curr_count > max_count){
                max_count = curr_count;
                z = data[i-1];
            }
            curr_count = 1;
        }
    }
    return z;
}
```

testbench

```cpp
int main(){
    vector<double> data;
    double item;

    string file1,file2;

    //Input data file
    cout << "Input file name: ";
    cin >> file1;
    ifstream fin(file1);
    ofstream fout(file2);

    while(fin >> item)
    {
        data.push_back(item);
    }
    fin.close();

    //Sort data file to another textfile
    cout << "Output file name: ";
    cin >> file2;
    fout.open(file2);

    double i,j,first,element;
    int n = data.size();

    for(i=n-1; i>0; i--)
    {
        first = 0;
        for(j=1; j<=i; j++)
        {
            if(data[j] > data[first])
            first = j;
        }
        //<Uncomment following three lines to sort data file in ascending order>
        //element = data[first];
        //data[first] = data[i];
        //data[i] = element;
    }

    for(double i=0; i<n; i++)
    {
        fout << data[i] << " " << "\n";
    }
    fout.close();

    double sum = accumulate(data.begin(), data.end(), 0.);
    double w = max(data,n);
    double x = mean(sum,n);
    double y = median(data,n);
    double z = mode(data,n);

    cout << "Number of Elements: " << n << "\n";
    cout << "Sum: " << sum << "\n";
    cout << "Maximum: " << w << "\n";
    cout << "Mean: " << "(" << sum << "/" << n << ")" << " = " << x << "\n";

    if(n % 2 == 0){
        cout << "Median: " << "(" << data[n/2 - 1] << "+" << data[n/2] << ")" << "/2 = " << y << "\n";
    }else{
        cout << "Median: " << y << "\n";
    }

    cout << "Mode: " << z << "\n";

    return 0;
}
```

```
edwin@ubuntu-edo:~/hls$ cd "/home/
Input file name: data.txt
Output file name: sort.txt
Number of Elements: 1600
Sum: 411349
Maximum: 1023
Mean: (411349/1600) = 257.093
Median: (59+104)/2 = 81.5
Mode: 61
edwin@ubuntu-edo:~/hls$
```

```
edwin@ubuntu-edo:~/hls/i++.prj/components$ ls
max  mean  median  mode
edwin@ubuntu-edo:~/hls/i++.prj/components$
```

Data Set (Simulation Data)

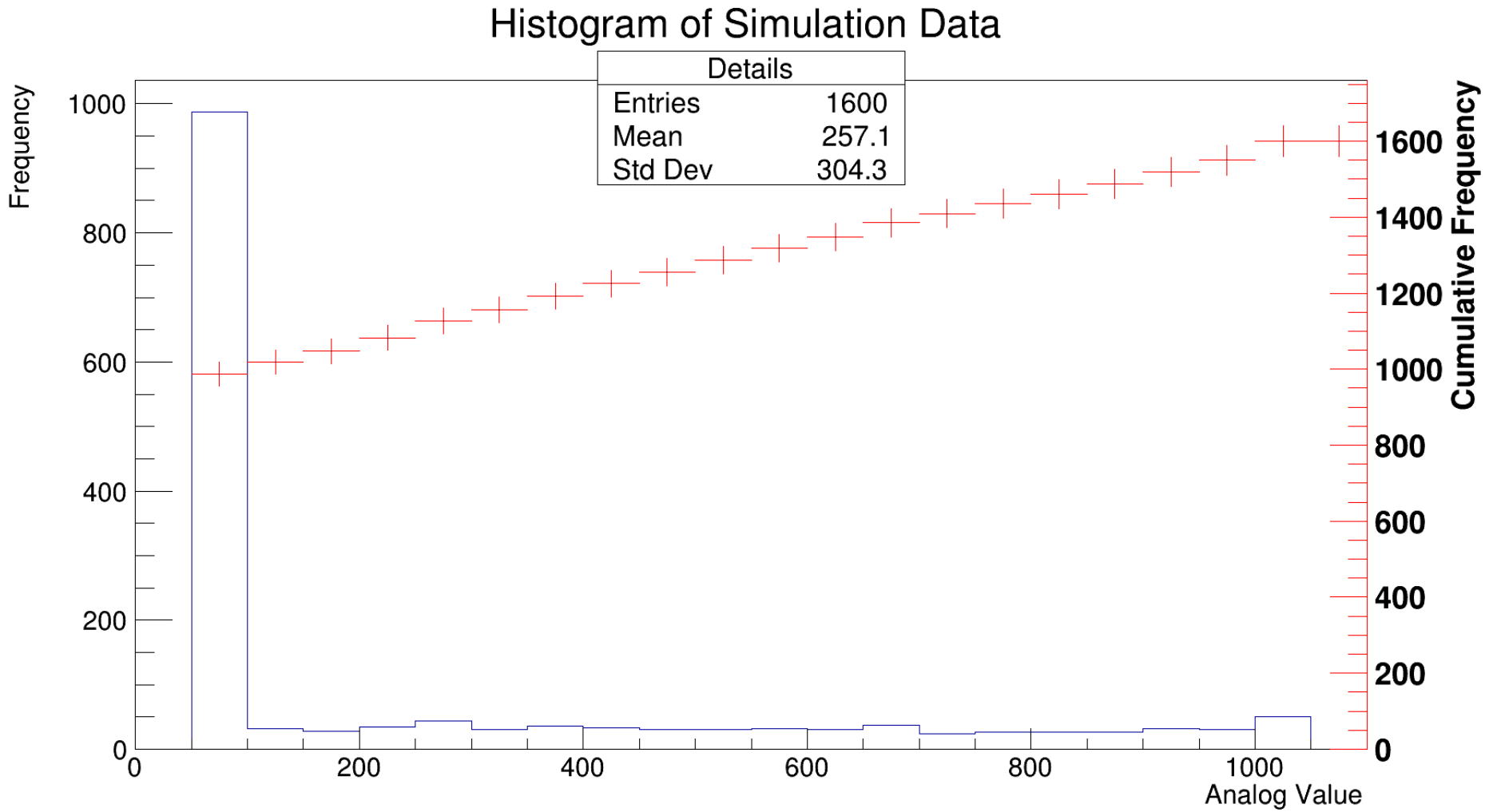
data.txt

```
61 60 60 59 60 60 937 60 61 1023 60 59 61 60 798 61 914 59 68 61 885 1001 85 61 61 945 60 59 378 750 59 161 61 245 61 61 907 245 59 59
60 636 61 59 59 61 60 60 264 568 948 187 1023 60 60 61 634 60 60 59 59 59 61 59 59 59 60 61 60 61 60 159 60 960 453 59 60 61 60 1023
59 443 227 60 60 60 484 606 61 61 61 1023 59 60 61 59 60 61 224 60 61 59 59 60 60 61 429 190 60 59 59 59 61 59 758 59 351 924 522 61
61 504 647 59 1023 59 61 60 60 437 60 60 60 60 896 61 61 60 60 711 106 60 988 60 61 143 59 112 59 59 61 59 61 60 60 59 59 258 59 680
203 61 60 59 643 848 816 1016 935 60 61 61 60 59 293 59 60 587 130 60 59 61 60 59 86 1001 61 60 59 61 312 797 61 291 128 61 850 694 230 501
59 689 466 60 60 61 876 61 59 796 60 60 841 59 61 61 61 60 791 60 728 147 985 115 237 61 61 61 60 60 59 109 177 705 61 59 561 415 1023 288
61 359 274 316 61 491 396 551 61 60 250 854 61 59 60 954 59 452 1023 60 966 61 314 59 157 60 60 766 589 699 59 588 60 437 60 61 822 820 61 90
61 439 494 134 61 59 59 59 693 61 61 435 60 60 60 60 929 843 802 1021 560 61 266 722 636 59 297 943 456 810 1023 339 620 61 60 623 188 753 61 61
381 974 60 61 60 60 60 60 61 659 59 59 907 714 59 198 498 60 61 59 59 200 483 878 234 60 59 595 955 395 59 970 59 71 61 60 60 416 61 296
697 60 125 61 1023 61 59 60 689 60 61 229 60 59 61 788 61 60 59 60 279 496 61 586 155 59 693 257 59 59 875 479 59 60 60 59 61 59 60 896
820 60 59 59 61 59 61 59 61 61 59 60 298 785 273 60 951 499 59 60 533 129 59 61 60 60 61 60 217 59 647 59 60 61 61 947 59 61 61 60
731 59 60 59 649 717 59 193 339 472 394 59 60 147 395 1023 1023 60 61 638 59 61 61 59 59 60 60 665 583 886 820 167 693 60 59 990 60 59 60 60
59 1016 61 150 853 608 60 586 61 61 60 61 61 59 825 455 61 59 929 254 352 60 60 61 61 571 61 59 439 59 682 652 60 341 200 549 620 60 60 60
61 389 61 59 59 432 278 59 60 59 60 60 60 273 580 59 802 976 838 90 678 1023 60 979 60 758 60 60 907 61 61 61 61 61 61 470 60 61 60 755
92 61 811 59 957 375 860 61 353 60 60 583 59 60 61 844 688 547 316 968 59 61 59 61 59 60 60 749 59 61 660 60 61 380 397 799 873 61 59 60
59 59 59 1023 779 59 59 61 59 120 60 61 1023 534 63 60 60 1015 744 870 662 554 942 59 61 502 59 59 381 60 61 60 621 60 59 951 61 61 59 60
61 60 61 663 59 1023 479 358 149 822 649 752 60 606 60 59 60 603 59 60 641 914 764 921 60 59 408 59 61 61 60 78 60 656 61 374 60 59 533 59
61 60 601 238 60 61 61 59 140 834 59 96 61 61 60 59 75 121 61 798 60 164 59 722 344 59 134 393 61 61 61 724 61 743 273 61 937 61 242 60
906 60 60 1003 61 59 60 453 60 61 61 59 60 61 59 59 60 86 61 61 60 553 60 59 60 59 60 707 231 463 60 597 59 59 533 60 59 312 60 61
60 235 59 484 380 477 61 61 59 60 61 59 61 61 60 61 777 59 59 60 61 273 59 60 1001 59 60 60 60 245 431 275 352 61 668 59 61 60 61 59
104 59 868 59 148 61 59 93 60 60 60 61 60 188 648 732 59 74 207 59 60 359 260 59 251 60 280 59 60 61 59 186 60 454 59 61 59 61 60 252
61 61 59 371 234 60 59 61 60 59 60 59 369 580 59 420 61 60 660 73 166 190 59 71 739 1023 59 61 61 383 993 60 232 59 61 59 404 216 534 60
60 61 61 1023 205 965 268 453 721 59 60 500 60 584 423 1023 60 102 60 59 60 192 61 60 108 59 59 61 59 864 60 59 61 59 939 59 414 1023 182 1023
767 61 280 899 325 61 541 61 61 61 60 61 59 59 61 478 59 704 851 141 61 60 61 841 61 60 60 61 504 61 301 60 996 861 809 568 60 60 895 59
225 293 512 155 347 60 59 61 988 59 79 476 61 832 995 1023 59 60 59 59 60 188 373 254 61 61 61 1023 61 59 60 61 377 60 61 60 304 60 61 59
61 59 59 59 61 61 60 411 61 61 60 60 61 61 61 400 61 59 61 61 513 59 59 1015 61 59 962 303 59 617 61 59 697 59 300 653 921 61 713 61
61 61 61 1023 60 61 109 60 986 61 59 61 59 60 60 60 60 61 60 555 60 59 61 60 430 59 343 61 60 679 60 577 59 122 59 585 59 60 61 61
476 420 293 314 60 422 61 479 59 92 59 360 61 59 60 59 60 59 59 906 59 259 61 59 448 60 60 59 60 60 643 60 136 950 286 59 60 59 60 908
60 61 61 126 331 860 622 61 59 59 61 133 259 61 61 327 788 725 845 60 61 61 204 240 61 224 934 78 60 60 200 61 59 273 61 59 59 61 152 178
313 60 61 395 59 59 515 59 60 60 61 1023 59 61 59 60 61 880 59 59 916 61 60 439 775 60 60 59 61 61 841 263 59 59 61 61 303 59 61 60
61 926 61 894 221 411 688 61 687 801 61 60 890 445 61 965 60 61 61 60 61 61 61 59 61 61 1007 771 159 61 61 60 59 485 1015 59 60 349 541 61
563 59 490 729 875 314 257 219 1016 59 60 539 61 59 61 299 510 60 60 60 1023 1023 775 60 60 397 61 60 59 913 59 60 809 312 60 314 60 59 59 61
60 59 60 60 60 598 218 569 59 722 317 59 59 61 549 59 962 60 60 684 239 59 637 59 61 380 61 918 59 59 1023 59 173 153 61 159 773 636 61 839
60 61 59 218 61 59 61 61 60 842 60 60 61 59 61 61 384 61 59 660 59 787 60 107 59 662 906 526 329 59 61 59 59 60 59 72 59 60 59 61
90 59 60 59 74 60 59 535 155 947 61 1023 1023 592 61 636 59 59 61 59 60 61 61 61 60 84 61 59 60 60 579 60 1023 427 59 60 60 746 131 60
61 504 61 59 61 671 60 59 1023 60 59 60 59 59 61 60 60 60 59 61 61 60 59 60 1023 60 59 1012 60 61 446 59 275 737 61 105 414 59 433 564
386 61 61 61 61 59 543 61 890 60 333 60 1023 59 919 276 61 60 256 60 59 59 59 59 974 61 59 60 1023 61 60 251 59 68 61 59 61 60 59 61
61 60 589 1023 60 60 61 59 249 467 965 60 534 295 61 60 77 61 448 59 60 60 299 344 542 60 59 651 202 61 1023 60 59 772 597 59 977 735 385 61
61 675 141 522 60 689 59 369 61 666 60 59 133 677 59 563 494 985 60 60 59 60 299 694 61 444 59 1023 60 514 61 913 59 59 60 994 61 429 352 60
394 621 61 60 922 543 315 60 266 60 61 60 640 59 60 60 633 60 59 60 60 613 60 61 61 230 885 431 61 59 59 979 60 692 59 501 61 61 60 61
```

sort.txt

```
61
60
60
59
60
60
937
60
61
1023
60
59
61
60
798
61
914
59
68
61
885
1001
85
61
61
945
60
59
378
750
59
161
61
245
61
61
907
245
59
59
```

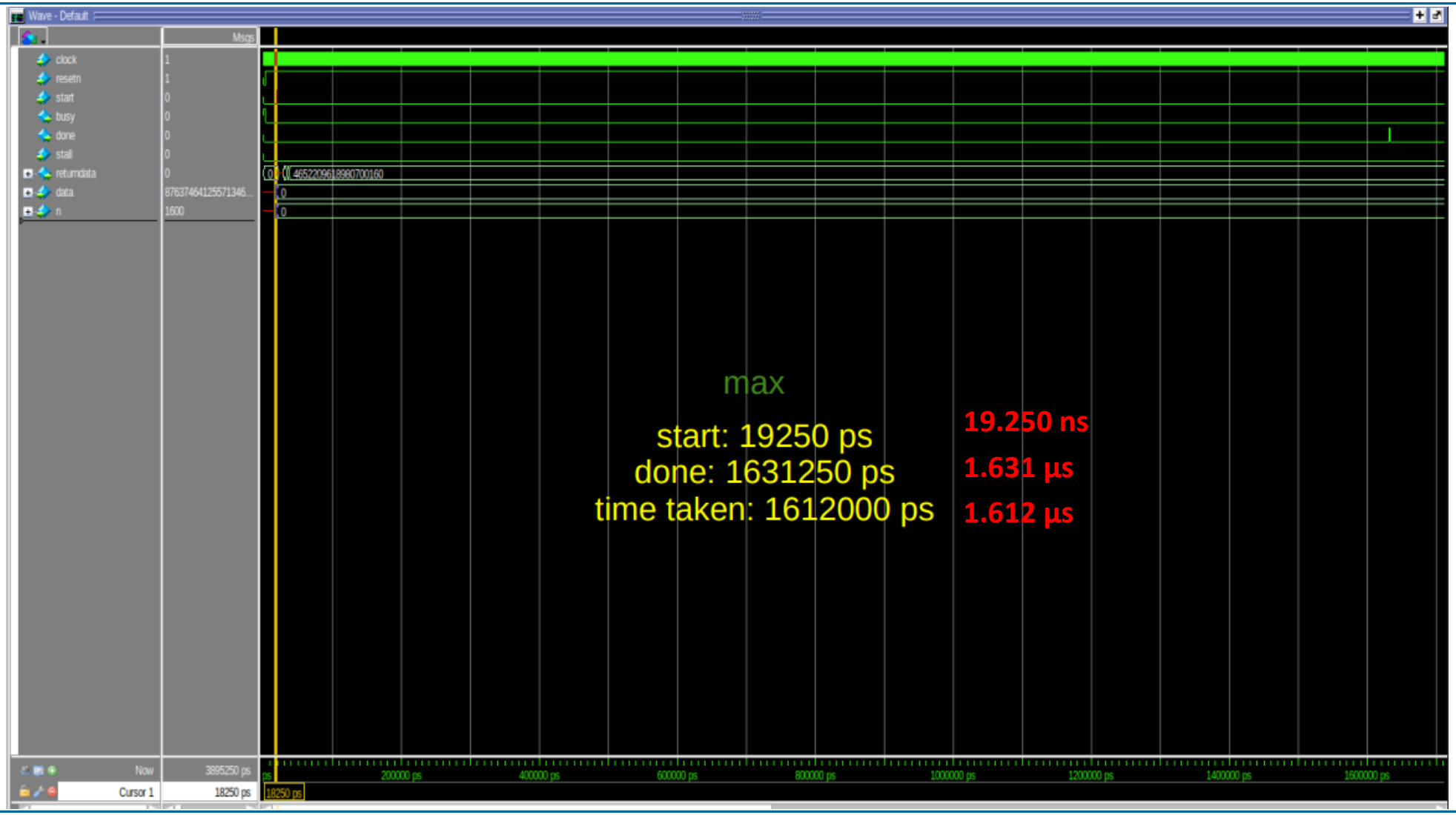
Histogram


Histogram of Simulation Data
Details
Entries 1600
Mean 257.1
Std Dev 304.3
Frequency
Cumulative Frequency
Analog Value
0
200
400
600
800
1000
1200
1400
1600

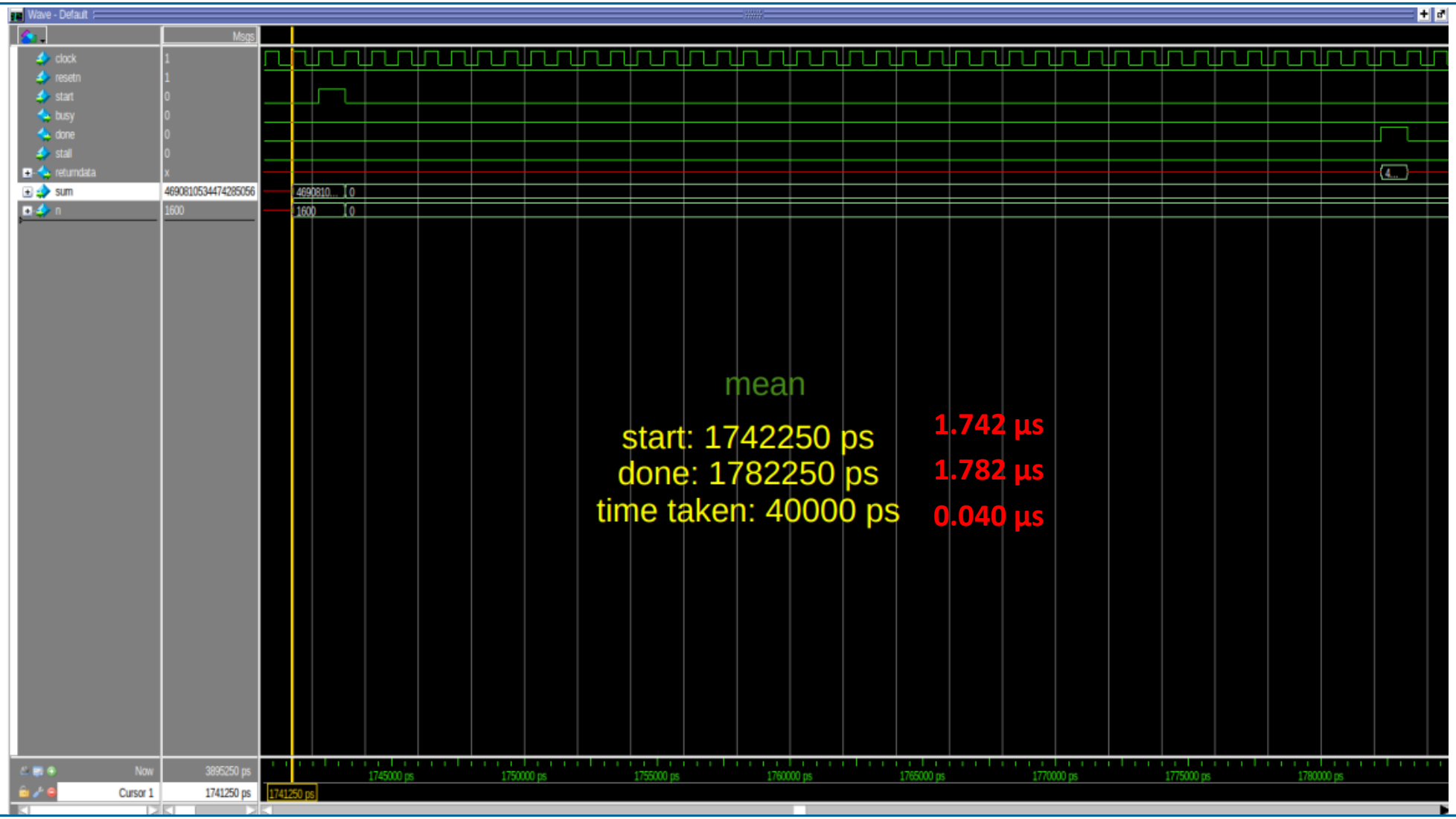
Waveform


Wave - Default
Msgs
clock
start
done
MAX
data
n
MEAN
sum
MEDIAN
MODE
Now
3895250 ps
500000 ps
1000000 ps
1500000 ps
2000000 ps
2500000 ps
3000000 ps
3500000 ps
4000000 ps

Waveform Max

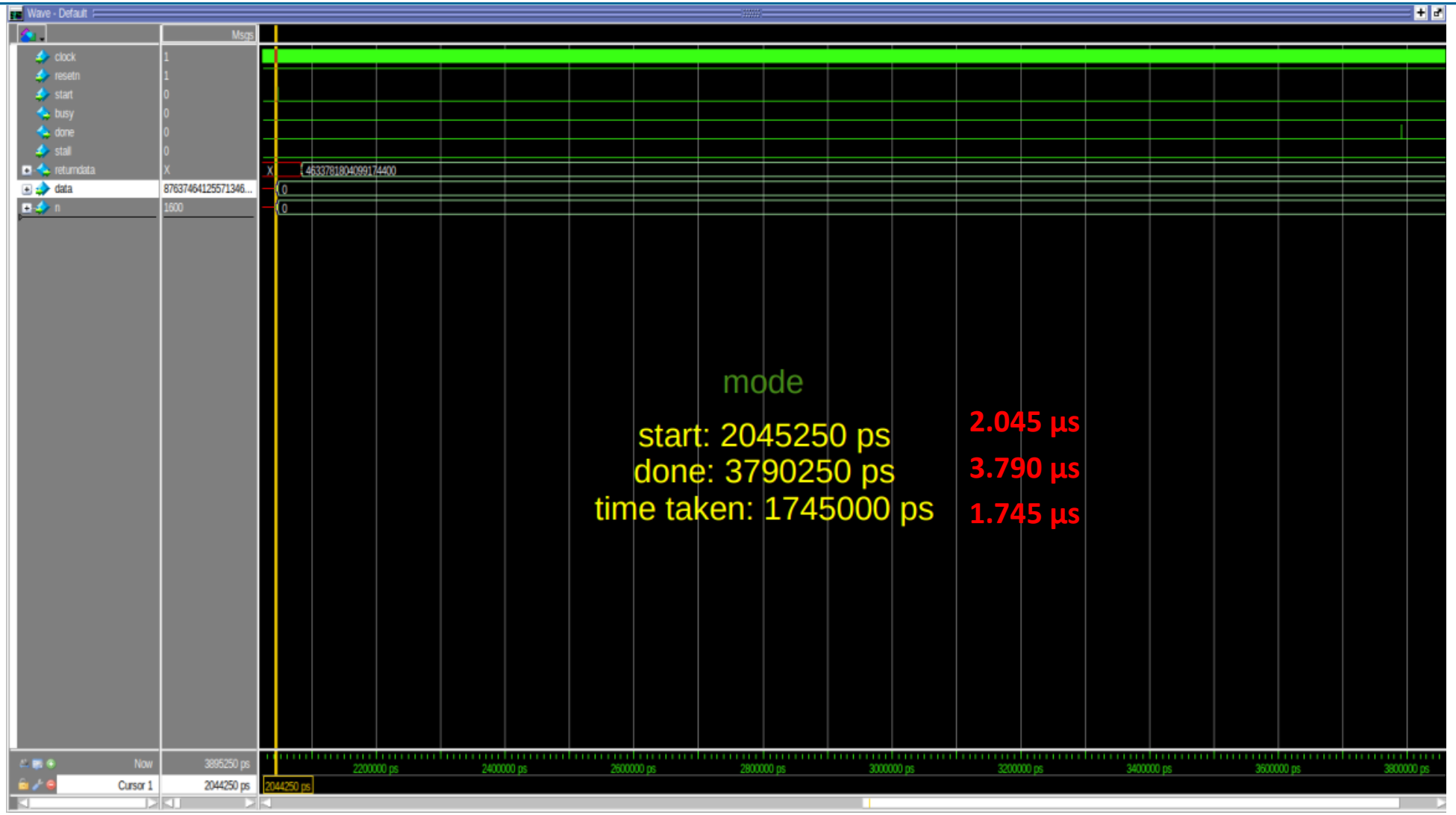


Waveform Mean



Waveform Median

Waveform Mode



Enqueue Function Calls

- Function calls to the simulator are sequential by default
- Allows component to be simulated in a pipelined fashion
  - Enqueued component invocations not run until `ihc_hls_component_run_all()` is invoked

```
ihc_hls_enqueue(ret_ptr, ptr_to_component_func, arguments,…)
```

  - Enqueues one invocation of an HLS component with return type

```
ihc_hls_enqueue_noret(ptr_to_component_func, arguments,…)
```

  - Enqueues one invocation of an HLS component with void return type

```
ihc_hls_component_run_all(component_func_name)
```

  - All enqueued invocations of component pushed into the component as quickly as possible

Non-enqueued vs Enqueued

enqx.cpp

C: > intelFPGA > hls > tutorials > usability > enq_no > enqx.cpp > ...

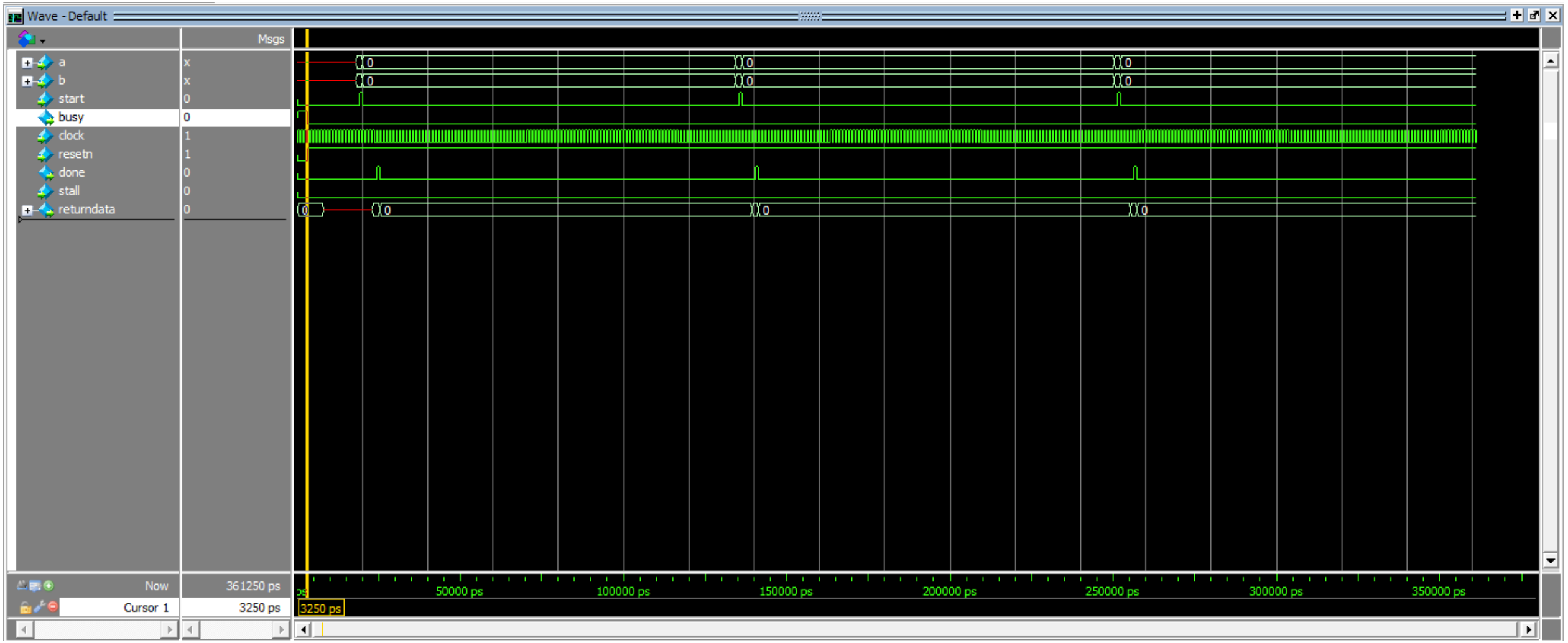
```
1   #include "HLS/hls.h"
2   #include <stdio.h>
3
4   component int dut(int a, int b) {
5     return a*b;
6   }
7   int main (void) {
8     int x1, x2, x3;
9
10    x1 = dut(1, 2);
11    x2 = dut(3, 4);
12    x3 = dut(5, 6);
13    printf("x1 = %d, x2 = %d, x3 = %d\n", x1, x2, x3);
14
15    return 0;
16  }
17
```

enq.cpp

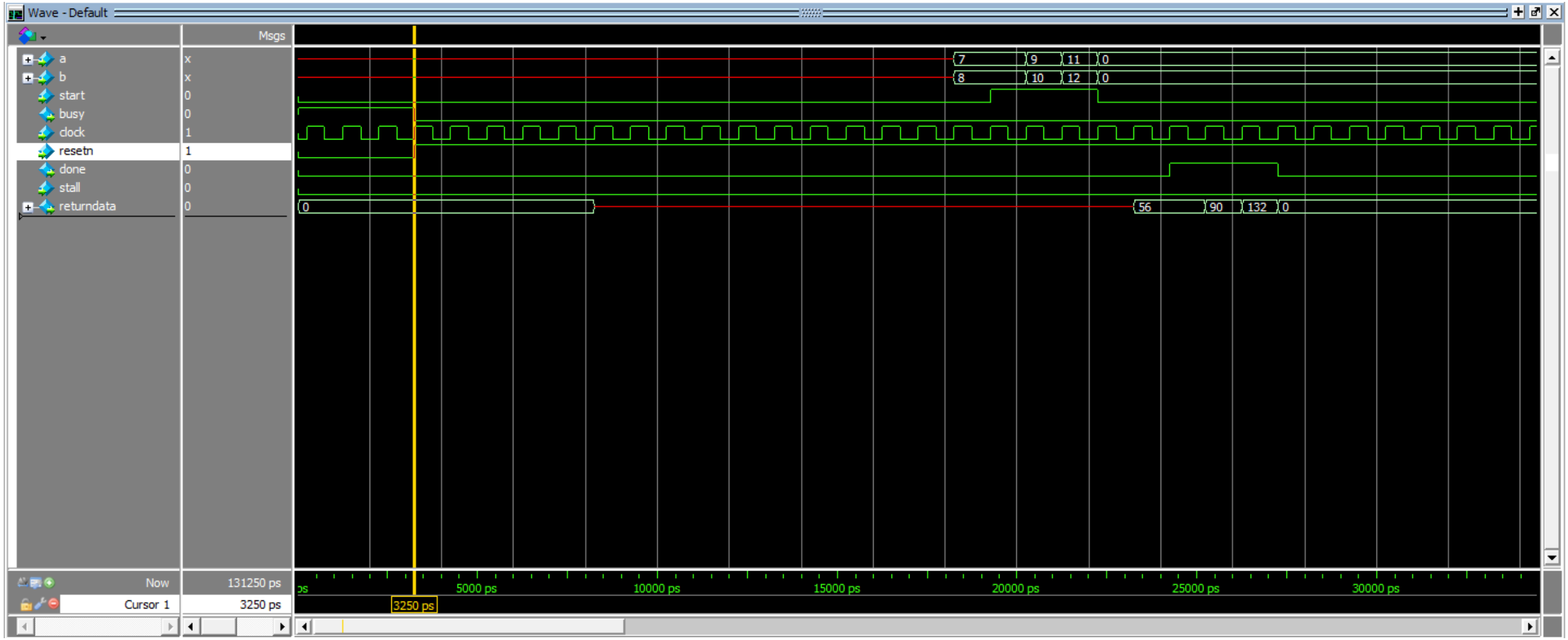
C: > intelFPGA > hls > tutorials > usability > enq > enq.cpp > ...

```
1   #include "HLS/hls.h"
2   #include <stdio.h>
3
4   component int dut(int a, int b) {
5     return a*b;
6   }
7   int main (void) {
8     int x1, x2, x3;
9
10    ihc_hls_enqueue(&x1, &dut,  7,  8);
11    ihc_hls_enqueue(&x2, &dut,  9, 10);
12    ihc_hls_enqueue(&x3, &dut, 11, 12);
13    ihc_hls_component_run_all(dut);
14
15    printf("x1 = %d, x2 = %d, x3 = %d\n", x1, x2, x3);
16
17    return 0;
18  }
```

Non-enqueued Waveform



Enqueued Waveform



Loops

part_1_sequential_loop.cpp

C: > intelFPGA > hls > tutorials > system_of_tasks > parallel_loop > part_1_sequential_loop.cpp > ...

```cpp
#include "HLS/hls.h"
#include <iostream>
#include <math.h>

constexpr int N_SUM_INTERVAL = 4;
constexpr int N_SAMPLES = 64;

component float add_mult(ihc::stream_in<float> &sig_in) {
  hls_memory float add_buffer[N_SAMPLES/N_SUM_INTERVAL];

  float add = 0.0f;
  for (int i = 0; i < N_SAMPLES; ++i){
    if (i % N_SUM_INTERVAL == 0) {
      add = 0.0f;
    }
    add += sig_in.read();
    if ( (i+1) % N_SUM_INTERVAL == 0) {
      add_buffer[i/N_SUM_INTERVAL] = add;
    }
  }

  float res = 1.0f;
  for (int i = 0; i < N_SAMPLES/N_SUM_INTERVAL; ++i){
    res *= add_buffer[i];
  }

  return res;
}

int main() {
  ihc::stream_in<float> sig_in;
  for (int i = 1; i <= N_SAMPLES; i++) {
    // Generate a simple sawtooth wave;
    float data = i * 0.05f;
    sig_in.write(data);
  }

  std::cout << "Testing sequential loops.\n";
  float res = add_mult(sig_in);
  std::cout << "Result = " << res <<"\n";
  if (fabs(res - 1.44091e+11) < (res * 1e-5)) {
    std::cout << "PASSED\n";
  } else {
    std::cout << "FAILED\n";
  }

  return 0;
}
```

part_2_merged_loop.cpp

C: > intelFPGA > hls > tutorials > system_of_tasks > parallel_loop > part_2_merged_loop.cpp > ...

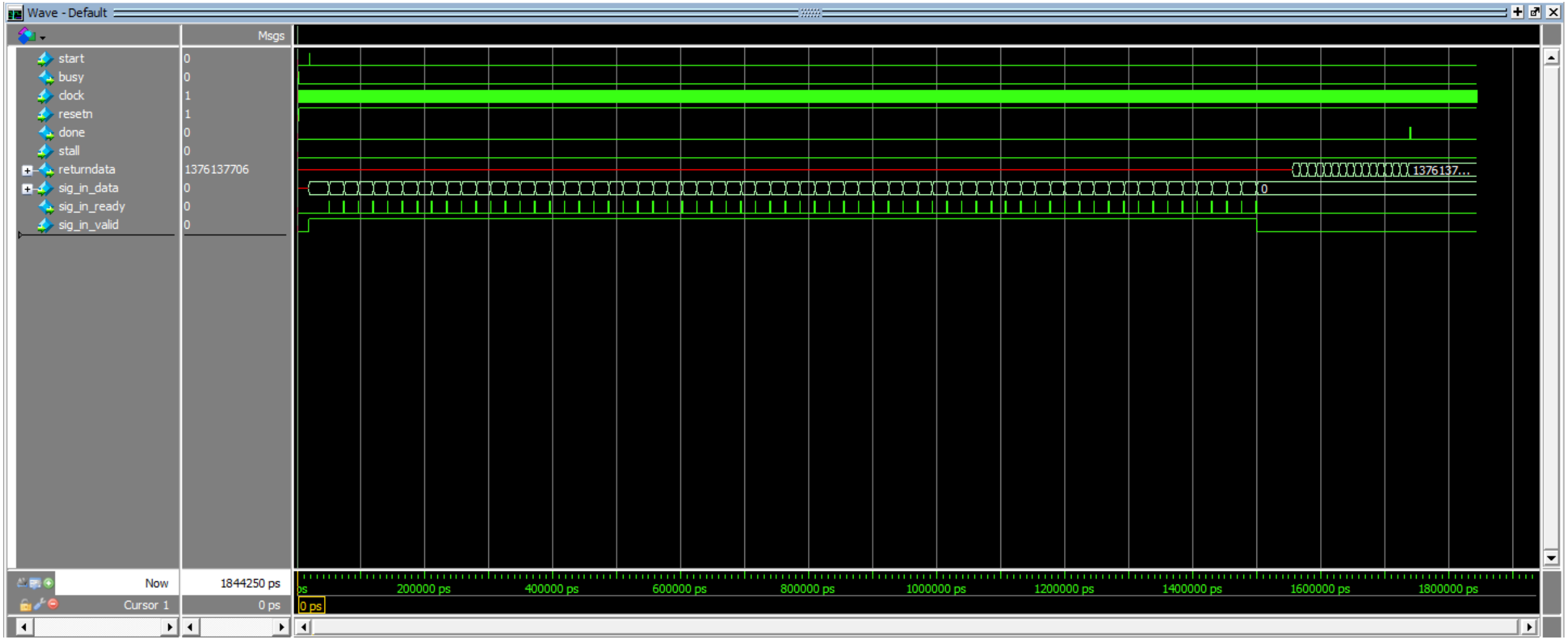
```cpp
#include "HLS/hls.h"
#include <iostream>
#include <math.h>

constexpr int N_SUM_INTERVAL = 4;
constexpr int N_SAMPLES = 64;

component float add_mult(ihc::stream_in<float> &sig_in) {
  float res = 1.0f;
  float add = 0.0f;
  for (int i = 0; i < N_SAMPLES; ++i){
    if (i % N_SUM_INTERVAL == 0) {
      add = 0.0f;
    }
    add += sig_in.read();
    if ( (i+1) % N_SUM_INTERVAL == 0) {
      res *= add;
    }
  }

  return res;
}

int main() {
  ihc::stream_in<float> sig_in;
  for (int i = 1; i <= N_SAMPLES; i++) {
    // Generate a simple sawtooth wave;
    float data = i * 0.05f;
    sig_in.write(data);
  }

  std::cout << "Testing merged loop.\n";
  float res = add_mult(sig_in);
  std::cout << "Result = " << res <<"\n";
  if (fabs(res - 1.44091e+11) < (res * 1e-5)) {
    std::cout << "PASSED\n";
  } else {
    std::cout << "FAILED\n";
  }

  return 0;
}
```

part_3_parallel_loop.cpp

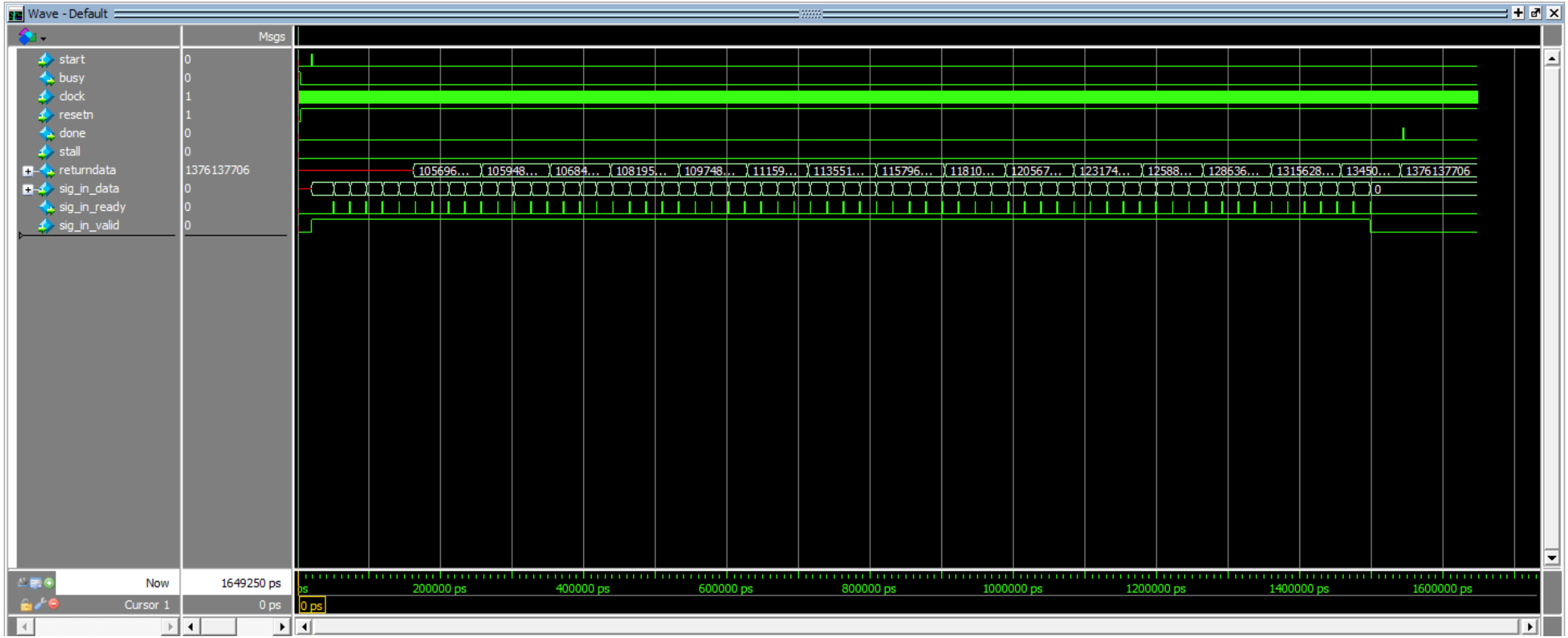
C: > intelFPGA > hls > tutorials > system_of_tasks > parallel_loop > part_3_parallel_loop.cpp > ...

```cpp
#include "HLS/hls.h"
#include <iostream>
#include <math.h>

constexpr int N_SUM_INTERVAL = 4;
constexpr int N_SAMPLES = 64;

ihc::stream<float> add_res;

float multiply_task(){
  float res = 1.0f;
  for (int i = 0; i < N_SAMPLES/N_SUM_INTERVAL; ++i){
    res *= add_res.read();
  }
  return res;
}

component float add_mult(ihc::stream_in<float> &sig_in) {
  ihc::launch<multiply_task>();
  float add = 0.0f;
  for (int i = 0; i < N_SAMPLES; ++i){
    if (i % N_SUM_INTERVAL == 0) {
      add = 0.0f;
    }
    add += sig_in.read();
    if ( (i+1) % N_SUM_INTERVAL == 0) {
      add_res.write(add);
    }
  }

  return ihc::collect<multiply_task>();
}

int main() {
  ihc::stream_in<float> sig_in;
  for (int i = 1; i <= N_SAMPLES; i++) {
    // Generate a simple sawtooth wave;
    float data = i * 0.05f;
    sig_in.write(data);
  }

  std::cout << "Testing parallel loops.\n";
  float res = add_mult(sig_in);
  std::cout << "Result = " << res <<"\n";
  if (fabs(res - 1.44091e+11) < (res * 1e-5)) {
    std::cout << "PASSED\n";
  } else {
    std::cout << "FAILED\n";
  }

  return 0;
}
```

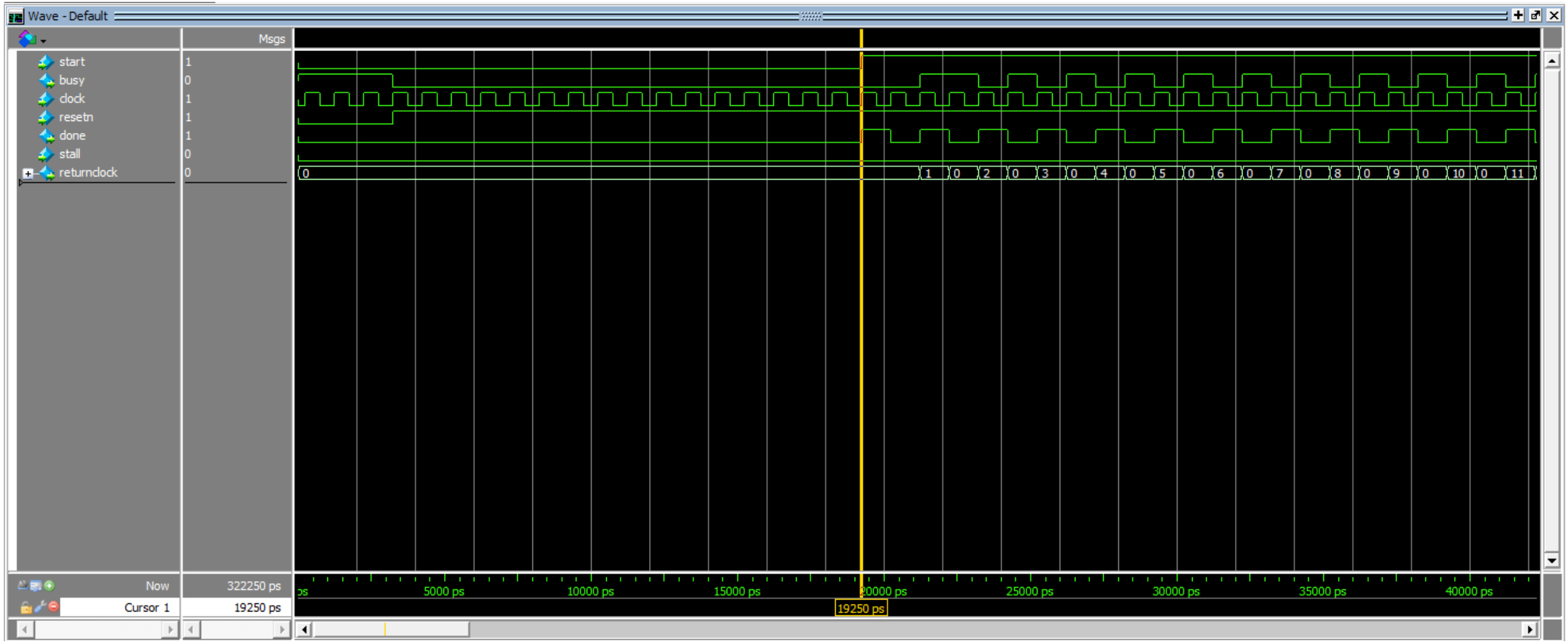
Sequential Loop



Parallel Loop



Clock Frequency -> 250MHz



i++ Options: FPGA Related Options

Option	Description
`--component <components>`	Specify a comma-separated list of function names to be synthesizes to RTL
`--clock <clock_spec>`	Optimizes the RTL for the specified clock frequency or period
`-ghdl`	Enable full debug visibility and logging of all signals when verification executable is run
`--quartus-compile`	Compiles the resulting HDL files using the Intel® Quartus® Prime software
`--simulator <simulator>`	Specify the simulator used for verification, "none" to skip testbench generation
`--x86-only`	Only create the executable for testbench, no RTL or cosim support
`--fpga-only`	Create FPGA component project, RTL and cosim support, no testbench binary

Example: `i++ -march=<fpga fam> --component mycomp --clock 400Mhz myfile.cpp`

There are many other optimization options available please see the *Intel HLS Compiler Reference Manual*

まとめ

- 普通のC++の知識だけでは足りない
- HLS化ための関数の理解は必要

きわめる。拓く。創り出す。

長崎総合科学大学

Nagasaki Institute of Applied Science

ご清聴ありがとうございました