Design Project 4: Simple Computer Assembly Language Programming

Jamahl Savage
Last Four Digits of Student ID: 3855

ECE 2504: Introduction to Computer Engineering

December 3, 2018

The Source Code: *Comments are in green

```
//  Starter source code program for ECE 2504

//  Project 4

//

// File: starter-program FINAL VER.txt

// Description: This text file implements the small assembly langauge program to process

// the values stored in an array in the data memory. The program uses two for loops to find the maximum

// and the minimum value in n length array

//

// The code will be used to display:

// 1. The location of the first array element (pointer to the array) (Data Memory Location: 00 )

// 2. The length of the array (Data Memory Location: 01 )

// 3. The Maximum value in the array (Data Memory Location: 02 )

// 4. The Minimum value in the array (Data Memory Location: 03 )

// 5. The Last four digits of my Student ID (Data Memory Location: 04 )

//

// The registers used in this program are used as follows:

// r0 - Mainly as the zero register. At times was used for controlling when the program needed to skip over
// parts of code.( Act like an if-else)

// r1 -  x (the for loop counter)

// r2 - y (the array pointer)

// r3 - the data memory pointer. Was also used to hold the result of some of the comparisons (brz
// statements)

// r4 - used to hold the current max or min

// r5 - used as the bit mask. Was also used to hold the result of whether the bits during a comparison
// were the same (brz statements)

// r6 - used to hold the next value in the array element

// r7 - was free to use for many purpose (No specific role)

//

// Created: 11/28/2018, Jamahl Savage, Virginia Tech

// Student ID Last Four Digits: 3855

////////////////////////////////////////////////////////////////////////////////////////////////////

ldi r0, 4

shl r0, r0
```

```
shl r0, r0

jmp r0

// The following set of load instructions read the final values of the variables in

// memory locations into r1-r5 so that we can see them on the LEDs.

// Your code must jump to this point after it has stored the results in data memory.

// You should jump to location 4.

// address 0x04

xor r0, r0, r0

ld r1, r0                   //r1<-M[0x00]

inc r0, r0

ld r2, r0                   //r2<-M[0x01]

inc r0, r0

ld r3, r0                   //r3<-M[0x02]

inc r0, r0

ld r4, r0                   //r4<-M[0x03]

inc r0, r0

ld r5, r0                   //r5<-M[0x04]

                            // Now loop forever

ldi r0, 0

                            //The address of this brz is the one used in validation: Address 0x0F.

brz r0, 0

                            // Your last instruction should be a jump to location 4

                            // in order to read the variables into registers r1-r5.

                            // address 0x10

//creating the last 4 digits of the student ID (3855) and storing them while using register 5 as a pointer to
memory location 4

ldi r6, 3

shl r6, r6

shl r6, r6

shl r6, r6

shl r6, r6

adi r6, r6, 7

adi r6, r6, 1               //this will now make it 8

shl r6, r6
```

```
shl r6, r6
shl r6, r6
shl r6, r6
adi r6, r6, 5
shl r6, r6
shl r6, r6
shl r6, r6
shl r6, r6
adi r6, r6, 5
ldi r3, 4                       //r3 points to the memory address 4, which the student id should be stored in
st r3, r6
//Create a zero register: r0
ldi r0, 0
//derive a for loop counter (x): r1
// derive a general DM pointer: r3
ldi r3, 1
ld r1, r3                       //r1 = x
dec r1, r1          //This allows us to start with the comparison of the current max and the next
                    // value without over iterating
//derive an array poiner(y): r2
ldi r3, 0
ld r2, r3                       //r2  = y
//derive a max and min: r4
ld r4, r2                       //register 4(the max holder) is initialized to be the first value in the array
//START OF THE FOR LOOP
inc r2, r2                      // i = 1
ld r6, r2                       //r6 = a[i] the next value in the array
//CREATE THE MASK USED FOR ONLY COMPARING THE SIGN OF THE FPs
ldi r5, 7
adi r5,r5, 1                    //Now is the value 8
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
```

```
adi r5, r5, 0
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 0
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 0         //the mask is now equal to 8000 in hex which is 16'b1000 0000 0000 0000
and r7, r4, r5   //Using a selective hold or mask to only maintain the sign bit of the current max. It
                 // will be held in r7 for later comparison
shr r7, r7       //We will have to shift the bit to the right one so that it is at location 15 of the 16 bit
                 // number (therefore there will only be positive values for comparing)
                 //r5 will no longer be need for bit masking at this point so we can use for holding the
                 // result of the bit masking of the next value in the array
and r5, r6, r5   //Using a selective hold or mask to only maintain the sign bit of the next value in
                 // the array. It will be held in r5 for later comparison
shr r5, r5       //We will also have to shift the bit to the right so that it is at location 15 of the 16 bit
                 // number (therefore there will only be positive values for comparing)
                 //Next we will check to make sure the max and the next value in the array DON'T have
                 // the same sign and the result of the comparison will be stored in r3
xor r3, r5, r7
                 //Now we use brz to determine whether the max and the next value in the array have the
                 // same sign
brz r3, 8        //IF they have the same sign as each other (r3 = 0) then we must continue checking the
                 // rest of the bits
                 //ELSE we will check whether the max or the next value in the array is greater through
                 // means of subtraction
sub r3, r7, r5
brn r3, 3        // If this fails then that means the current max had a 1 as it's sign, thus meaning it was
                 // negative. Therefore not the max.
add r4, r0, r6   // This means that brn failed and that the next value in the array was the one with a 0 as
                 // its sign, thus meaning it was positive.
brz r0, 2        // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                 // INCREMENTING AND DECREMENTING
```

```
add r4, r0, r4     // If brn passes that means that the current max had a 0 as its sign, thus meaning it was
                   // positive. There the current max remains the max

brz r0, 30         // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                   // INCREMENTING AND DECREMENTING

brz r0,   -30      // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                   // THE LOOP

//THIS WILL MARK THE END OF CHECKING THE SIGNS OF THE MAX AND THE NEXT VALUE IN
//THE ARRAY. WE WILL NOW PROCEED TO CHECKING THE 4 BITS OF THE EXPONENT OF THE
//CURRENT MAX AND THE NEXT VALUE IN THE ARRAY IF NECESSARY

ldi r0, 1

ldi r5, 7

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 7

adi r5, r5, 1            //Now the value is 8

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 0

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 0            //the mask is now equal to 7800 in hex which is 16'b0111 1000 0000 0000

and r7, r4, r5     //Using a selective hold or mask to only maintain the exponent bits of the current max. It
                   // will be held in r7 for later comparison

                   //r5 will no longer be need for bit masking at this point so we can use for maintaining the
                   // result of the bit masking of the next value in the array

and r5, r6, r5     //Using a selective hold or mask to only maintain the exponent bits of the next value in
                   // the array. It will be held in r5 for later comparison

                   //Next we will check to make sure the max and the next value in the array DON'T have
                   // the same exponent bits and the result of the comparison will be stored in r3

xor r3, r5, r7
```

```
                //Now we use brz to determine whether the max and the next value in the array have the
                // same exponent bits

brz r3, 9       //IF they have the same exponent bits as each other (r3 = 0) then we must continue
                // checking the rest of the bits

                //ELSE we will check whether the max or the next value in the array is greater through
                // means of subtraction

ldi r0, 0

sub r3, r7, r5

brn r3, 3       //If this fails then that means the current max has a larger exponent thus it stays as the
                // max

add r4, r0, r4  //The max shall stay the new max

brz r0, 2       // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

add r4, r0, r6  //If brn passed then that means the next value in the array had a larger exponent and
                // thus should become the max

brz r0, 3       // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

brz r0,   -30   // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                // THE LOOP

//THIS WILL MARK THE END OF CHECKING THE 4 BITS OF THE EXPONENT OF THE MAX AND
//THE NEXT VALUE IN THE ARRAY. WE WILL NOW PROCEED TO CHECKING THE 11 BITS OF THE
//MANTISSA OF THE CURRENT MAX AND THE NEXT VALUE IN THE ARRAY IF NECESSARY

ldi r0, 1

brz r0, 30      // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

brz r0, -3      // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                // THE LOOP

ldi r5, 0

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 7

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 7
```

```
adi r5, r5, 7

adi r5, r5, 1                 //Now the value is 15

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 7

adi r5, r5, 7

adi r5,   r5, 1               //Now the value 15

                              //the mask is now equal to 07FF in hex which is 16'b0000 0111 1111 1111

and r7, r4, r5                //Using a selective hold or mask to only maintain the mantissa bits of the current
                              // max. It will be held in r7 for later comparison

                              //r5 will no longer be need for bit masking at this point so we can use for
                              // maintaining the result of the bit masking of the next value in the array

and r5, r6, r5                //Using a selective hold or mask to only maintain the mantissa bits of the next
                              // value in the array. It will be held in r5 for later comparison

                //Next we will check to make sure the max and the next value in the array DON'T have
                // the same mantissa bits and the result of the comparison will be stored in r3

xor r3, r5, r7

                //Now we use brz to determine whether the max and the next value in the array have the
                // same mantissa bits

brz r3, 9       //IF they have the same mantissa bits as each other (r3 = 0) then the current max and the
                // next value in the array must be the same, thus we can just maintain the same max

                //ELSE we will check whether the max or the next value in the array is greater through
                // means of subtraction

ldi r0, 0

sub r3, r7, r5

brn r3, 3       //If this fails then that means the current max has a larger mantissa thus it remains the
                // max

add r4, r0, r4  //The current max shall remain as the max

brz r0, 4       // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

add r4, r0, r6  //If brn passed then that means the next value in the array had a larger mantissa and thus
                // should be made the new max

brz r0, 2       // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

brz r0,   -32   // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                // THE LOOP
```

```
//THIS WILL MARK THE END OF COMPARISON ALGORITHM.

                    //Decrement the counter to indicate that I have completed an iteration also increment
                    // second counter (register 6) for going through the values in the array

inc r2, r2                    //ptr++ (array pointer++)

dec r1, r1                    //loop counter--

//END OF THE LOOP!!!!!!!!!!!!!!!!!!!!!!!!!!!

                    //Check the value of the counter register to see if it's done with the for loop. If it's done go
                    // 2 spots forward in the instructions and continue,

                    //else move forward once in the instructions (brz r0 will cause it to repeat the loop)

brz r1, 2

brz r0, -4          //DON'T FORGET TO SET THE OFFSET SO THAT IT REPEATS THE COMPARISON
                    // ALGORITHM IF THE LOOP IS NOT DONE!!!!!!

//The following will occur after the for loop has ended

//Now we store the max (register 4) into its appropriate location in data memory

ldi r3, 2

st r3, r4


////////////////////NOW TO CHECK FOR THE MIN////////////////////////////////////////////////////////////////////////////////


//derive a for loop counter (x): r1

// derive a general DM pointer: r3

ldi r3, 1

ld r1, r3                     //r1 = x

dec r1, r1                    //This allows us to start with the comparison of the current min and the
                              // next value without over iterating

//derive an array poiner(y): r2

ldi r3, 0

ld r2, r3                     //r2  = y

//derive a max and min: r4

ld r4, r2                     //register 4(the max holder) is initialized to be the first value in the array


//START OF THE FOR LOOP

inc r2, r2                    // i = 1

ld r6, r2                     //r6 = a[i] the next value in the array
```

```
//CREATE THE MASK USED FOR ONLY COMPARING THE SIGN OF THE FPs////////////////////////

ldi r5, 7
adi r5,r5, 1            //Now is the value 8
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 0
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 0
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 0           //the mask is now equal to 8000 in hex which is 16'b1000 0000 0000 0000
and r7, r4, r5          //Using a selective hold or mask to only maintain the sign bit of the current min. It
                        // will be held in r7 for later comparison
shr r7, r7      //We will have to shift the bit to the right one so that it is at location 15 of the 16 bit
                // number (therefore there will only be positive values for comparing)

                //r5 will no longer be need for bit masking at this point so we can use for holding the
                // result of the bit masking of the next value in the array
and r5, r6, r5  //Using a selective hold or mask to only maintain the sign bit of the next value in the
                // array. It will be held in r5 for later comparison
shr r5, r5      //We will have to shift the bit to the right one so that it is at location 15 of the 16 bit
                // number (therefore there will only be positive values for comparing)

                //Next we will check to make sure the min and the next value in the array DON'T have the
                // same sign and the result of the comparison will be stored in r3
xor r3, r5, r7

                //Now we use brz to determine whether the min and the next value in the array have the
                // same sign
brz r3, 8       //IF they have the same sign as each other (r3 = 0) then we must continue checking the
                // rest of the bits
```

```
                //ELSE we will check whether the min or the next value in the array is lesser through
                // means of subtraction

sub r3, r7, r5

brn r3, 3       // If this fails then that means the current min had a 1 as it's sign, thus meaning it was
                // negative. Therefore, it is the min.

add r4, r0, r4  // This means that brn failed and that the next value in the array was the one with a 0 as
                // its sign, thus meaning it was positive.

brz r0, 2       // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

add r4, r0, r6  // If brn passes that means that the current min had a 0 as its sign, thus meaning it was
                // positive. Therefore, the next value in the array is the new min

brz r0, 30      // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                // INCREMENTING AND DECREMENTING

brz r0,   -30   // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                // THE LOOP

//THIS WILL MARK THE END OF CHECKING THE SIGNS OF THE MIN AND THE NEXT VALUE IN
//THE ARRAY. WE WILL NOW PROCEED TO CHECKING THE 4 BITS OF THE EXPONENT OF THE
//CURRENT MIN AND THE NEXT VALUE IN THE ARRAY IF NECESSARY

ldi r0, 1

ldi r5, 7

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 7

adi r5, r5, 1           //Now the value is 8

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 0

shl r5, r5

shl r5, r5

shl r5, r5

shl r5, r5

adi r5, r5, 0           //the mask is now equal to 7800 in hex which is 16'b0111 1000 0000 0000
```

```
and r7, r4, r5     //Using a selective hold or mask to only maintain the exponent bits of the current min. It
                   // will be held in r7 for later comparison

                   //r5 will no longer be need for bit masking at this point so we can use for maintaining the
                   // result of the bit masking of the next value in the array

and r5, r6, r5     //Using a selective hold or mask to only maintain the exponent bits of the next value in
                   // the array. It will be held in r5 for later comparison

                   //Next we will check to make sure the min and the next value in the array DON'T have the
                   // same exponent bits and the result of the comparison will be stored in r3

xor r3, r5, r7

                   //Now we use brz to determine whether the min and the next value in the array have the
                   // same exponent bits

brz r3, 9          //IF they have the same exponent bits as each other (r3 = 0) then we must continue
                   // checking the rest of the bits

                   //ELSE we will check whether the min or the next value in the array is lesser through
                   // means of subtraction

ldi r0, 0

sub r3, r7, r5

brn r3, 3          // If this fails then that means the current min has a smaller exponent thus it is the min

add r4, r0, r4     //The current min shall remain the min

brz r0, 2          // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                   // INCREMENTING AND DECREMENTING

add r4, r0, r6     //If brn passed then that means the next value in the array had a smaller exponent and
                   // thus it should be the new min

brz r0, 3          // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                   // INCREMENTING AND DECREMENTING

brz r0,   -30      // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                   // INCREMENTING AND DECREMENTING

//THIS WILL MARK THE END OF CHECKING THE 4 BITS OF THE EXPONENT OF THE MIN AND THE
//NEXT VALUE IN THE ARRAY. WE WILL NOW PROCEED TO CHECKING THE 11 BITS OF THE
//MANTISSA OF THE CURRENT MIN AND THE NEXT VALUE IN THE ARRAY IF NECESSARY

ldi r0, 1

brz r0, 30         // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                   // INCREMENTING AND DECREMENTING

brz r0, -3         // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                   // THE LOOP

ldi r5, 0

shl r5, r5

shl r5, r5

shl r5, r5
```

```
shl r5, r5
adi r5, r5, 7
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 7
adi r5, r5, 7
adi r5, r5, 1           //Now the value is 15
shl r5, r5
shl r5, r5
shl r5, r5
shl r5, r5
adi r5, r5, 7
adi r5, r5, 7
adi r5,   r5, 1         //Now the value is 15
                        //the mask is now equal to 07FF in hex which is  16'b0000 0111 1111 1111
and r7, r4, r5          //Using a selective hold or mask to only maintain the mantissa bits of the current
                        // min. It will be held in r7 for later comparison
                //r5 will no longer be need for bit masking at this point so we can use for
                // maintaining the result of the bit masking of the next value in the array
and r5, r6, r5  //Using a selective hold or mask to only maintain the mantissa bits of the next value in the
                // array. It will be held in r5 for later comparison
                //Next we will check to make sure the min and the next value in the array DON'T have the
                // same mantissa bits and the result of the comparison will be stored in r3
xor r3, r5, r7
                //Now we use brz to determine whether the min and the next value in the array have the
                // same mantissa bits
brz r3, 9       //IF they have the same mantissa bits as each other (r3 = 0) then the current min and the
                // next value in the array must be the same, Thus we can just maintain the same min
                //ELSE we will check whether the min or the next value in the array is lesser through
                // means of subtraction
ldi r0, 0
sub r3, r7, r5
brn r3, 3       //If this fails then that means the current min has a larger mantissa thus it should NOT
                // stay the min
```

```
add r4, r0, r6    //The current min is replaced by the next value in the array as the new min

brz r0, 4         // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                  // INCREMENTING AND DECREMENTING

add r4, r0, r4    //If brn passed then that means the next value in the array had a larger mantissa and thus
                  // the current min REMAINS as the min

brz r0, 2         // TO THE END OF ALL THE IF ELSE STATEMENTS SO THAT SKIPS TO THE
                  // INCREMENTING AND DECREMENTING

brz r0,   -32     // TO THE BEGINNING OF ALL THE IF ELSE STATEMENTS SO THAT IT CONTINUES
                  // THE LOOP

//THIS WILL MARK THE END OF COMPARISON ALGORITHM.

                  //Decrement the counter to indicate that I have completed an iteration also increment
                  // second counter (register 6) for going through the values in the array

inc r2, r2                //ptr++ (array pointer++)

dec r1, r1                //loop counter--

                  //END OF THE LOOP!!!!!!!!!!!!!!!!!!!!!!!!!!

                  //Check the value of the counter register to see if it's done with the for loop. If it's done go
                  // 2 spots forward in the instructions and continue, else move forward once in the
                  // instructions (brz r0 will cause it to repeat the loop)

brz r1, 2

brz r0, -4        //DON'T FORGET TO SET THE OFFSET SO THAT IT REPEATS THE COMPARISON
                  // ALGORITHM IF THE LOOP IS NOT DONE!!!!!!

//The following will occur after the for loop has ended

//Now we store the max (register 4) into its appropriate location in data memory

ldi r3, 3

st r3, r4

//Give the register 0 the location in the instructions (Stating from 0 aka the top of the instructions)

ldi r0, 4

jmp r0            // to jump back to where the final data will be stored in the specified registers for viewing
                  // on the DE0 Nano Board
```

Test 1:

Floating Point values used in the test and their decimal values:

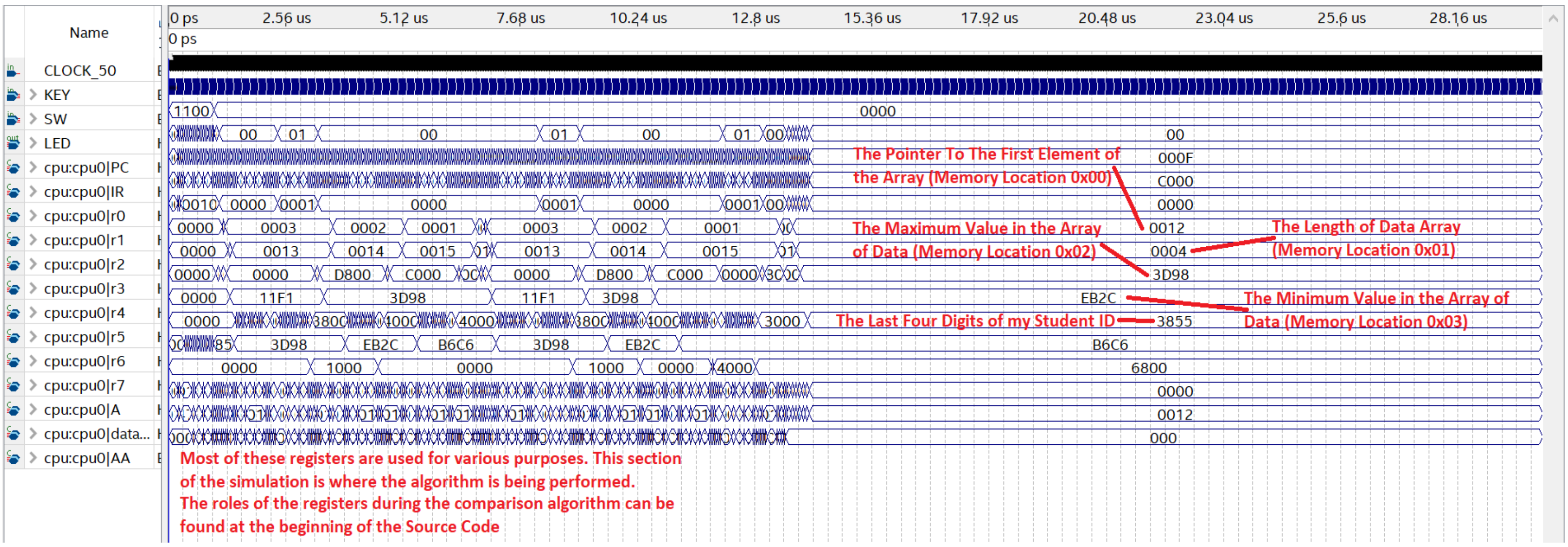
11F1: 0.038833618

3D98: 1.69921875 (This should be the Maximum Value)

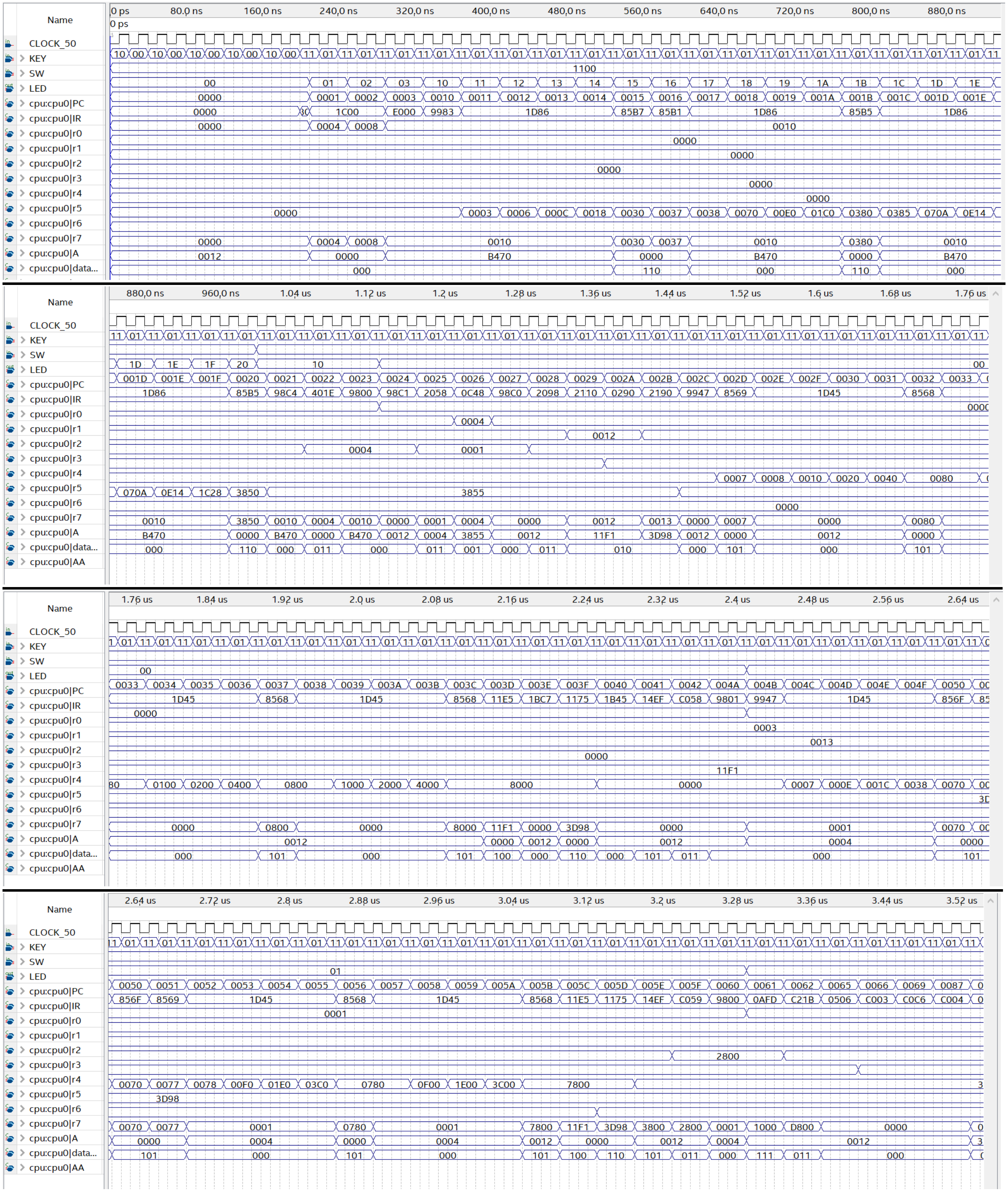
EB2C: -89.375 (This should be the Minimum Value)

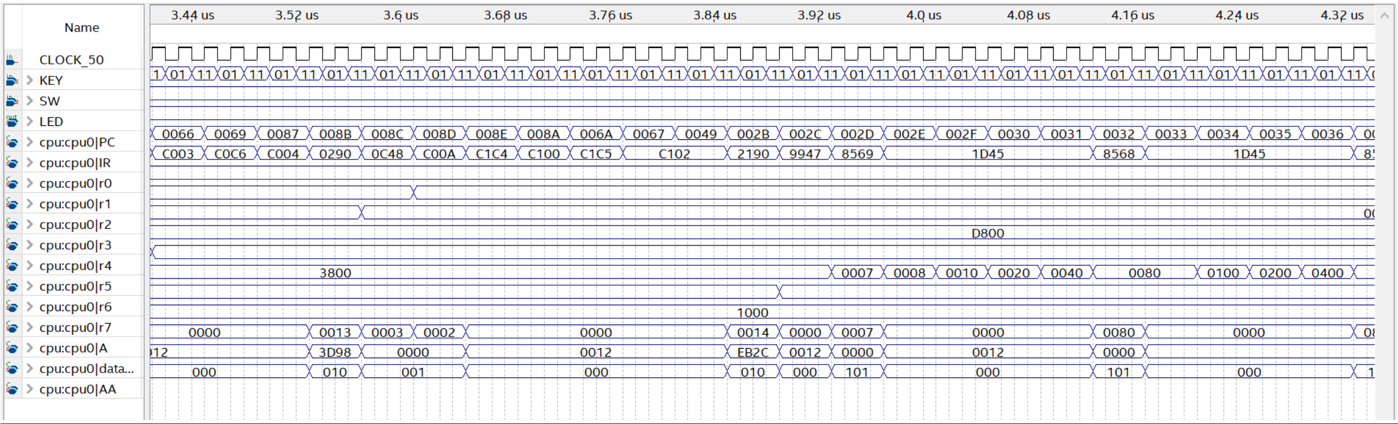
B6C6: -0.923339844

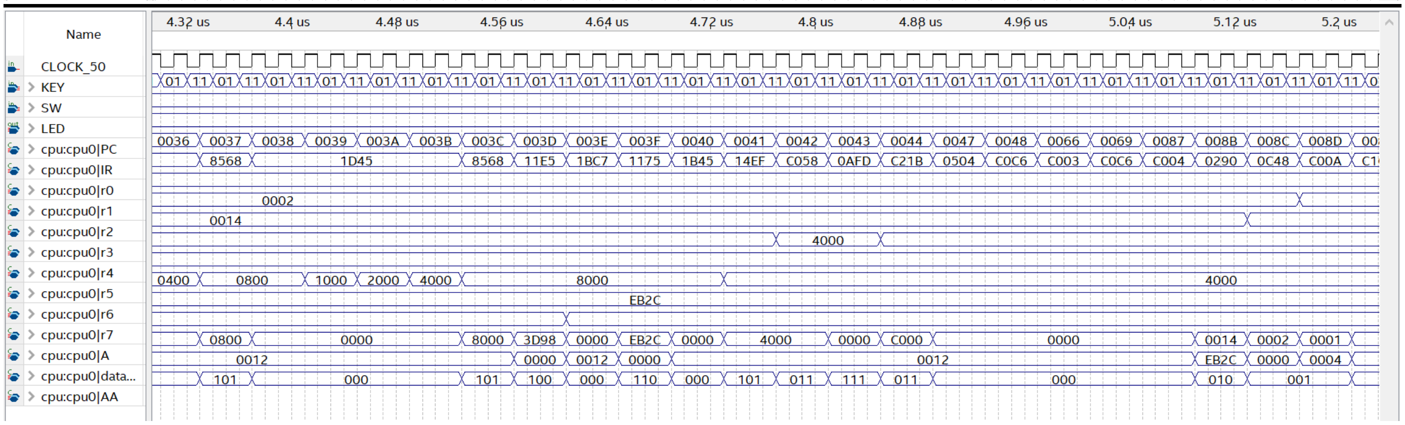
Full Simulation Results:

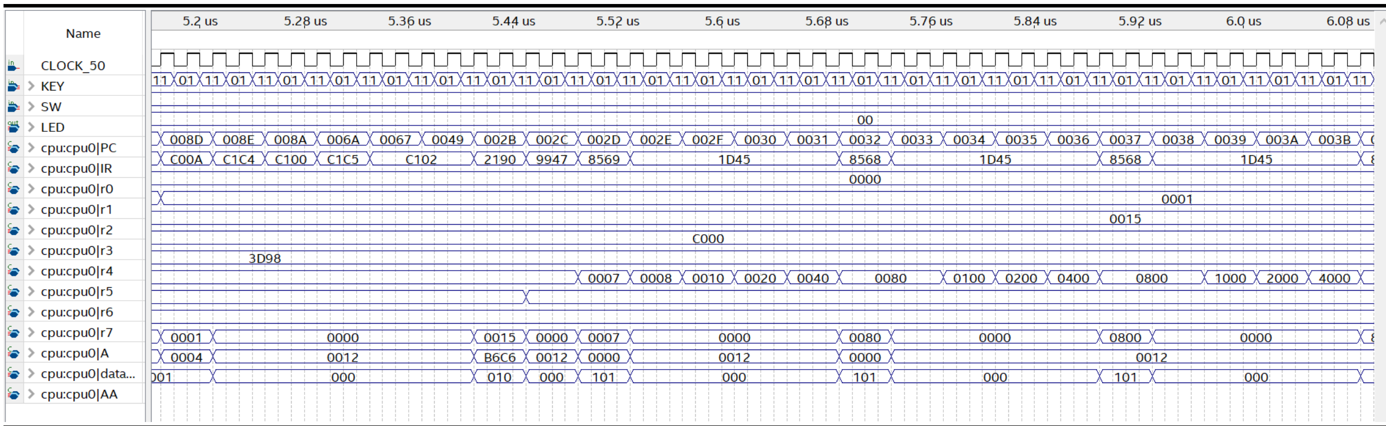


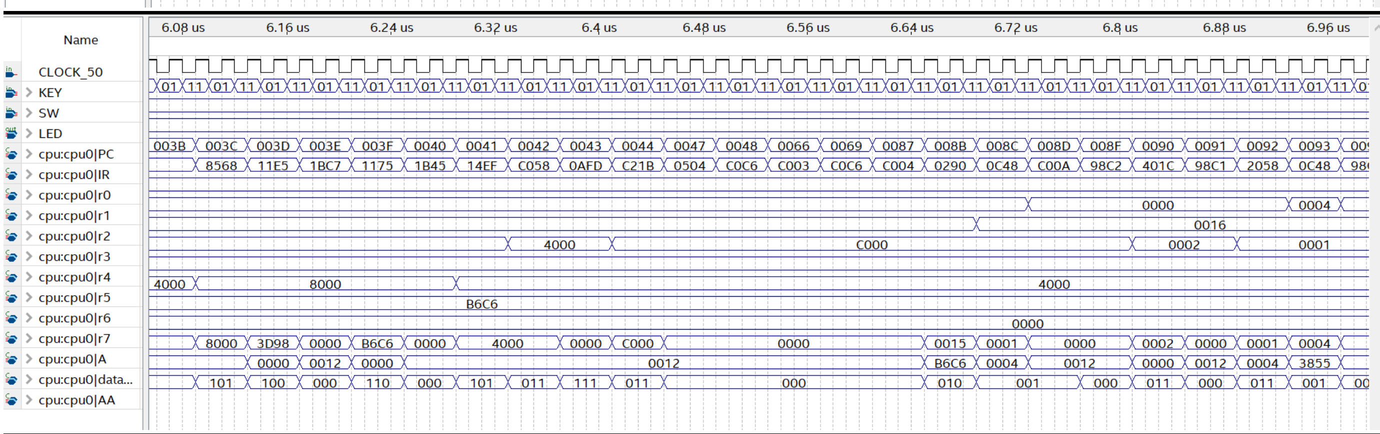
Test 1 Zoomed in Until Final Results of Simulation Are Generated:

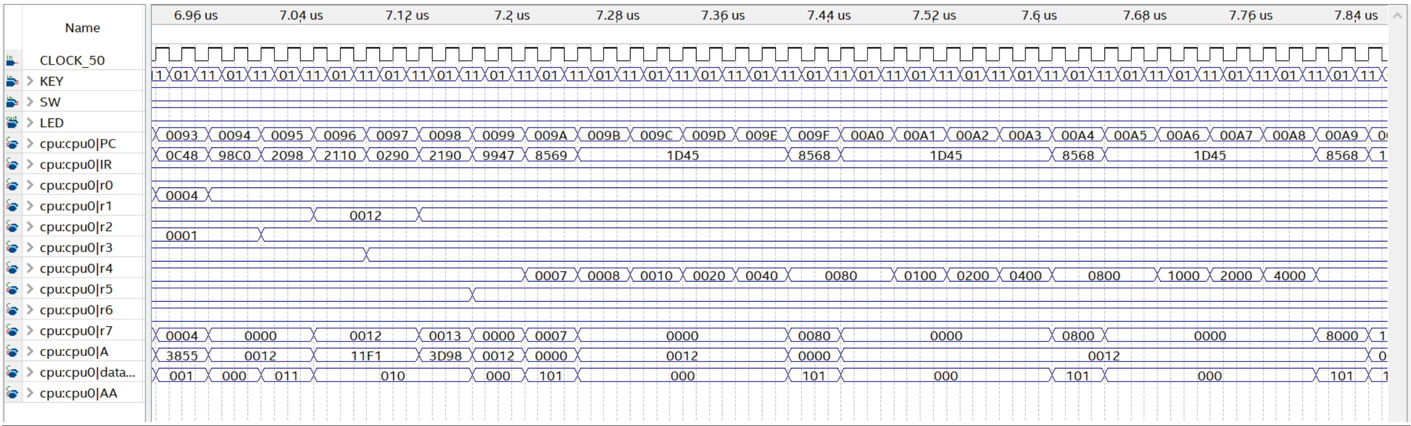


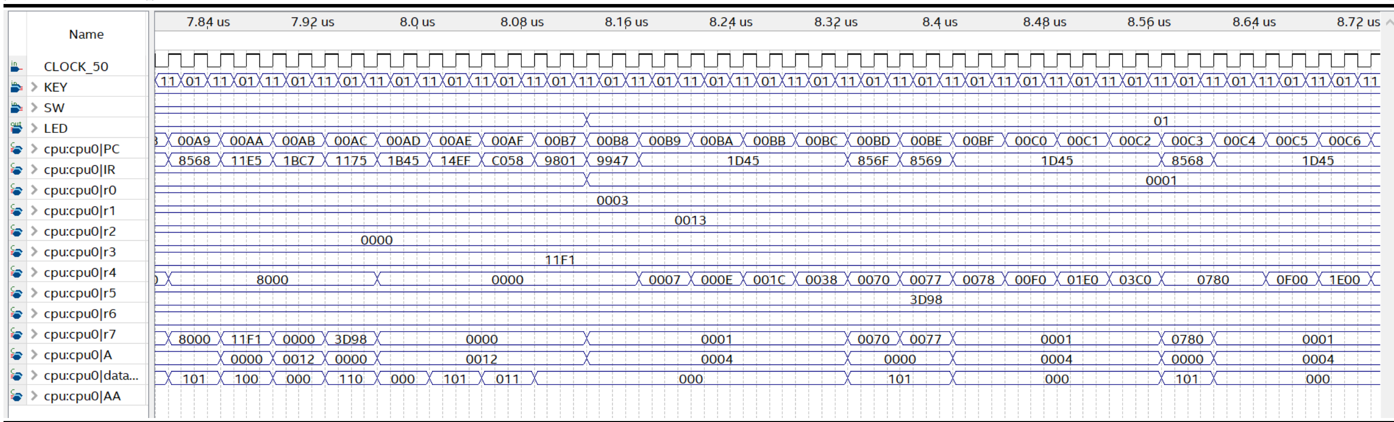

Name
3.44 us 3.52 us 3.6 us 3.68 us 3.76 us 3.84 us 3.92 us 4.0 us 4.08 us 4.16 us 4.24 us 4.32 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

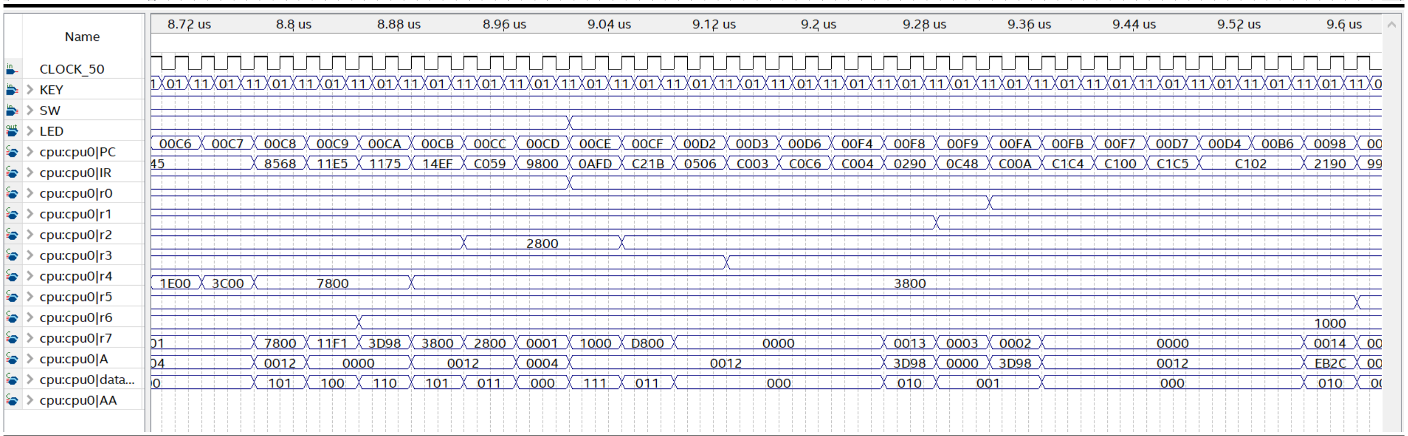

Name
4.32 us 4.4 us 4.48 us 4.56 us 4.64 us 4.72 us 4.8 us 4.88 us 4.96 us 5.04 us 5.12 us 5.2 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

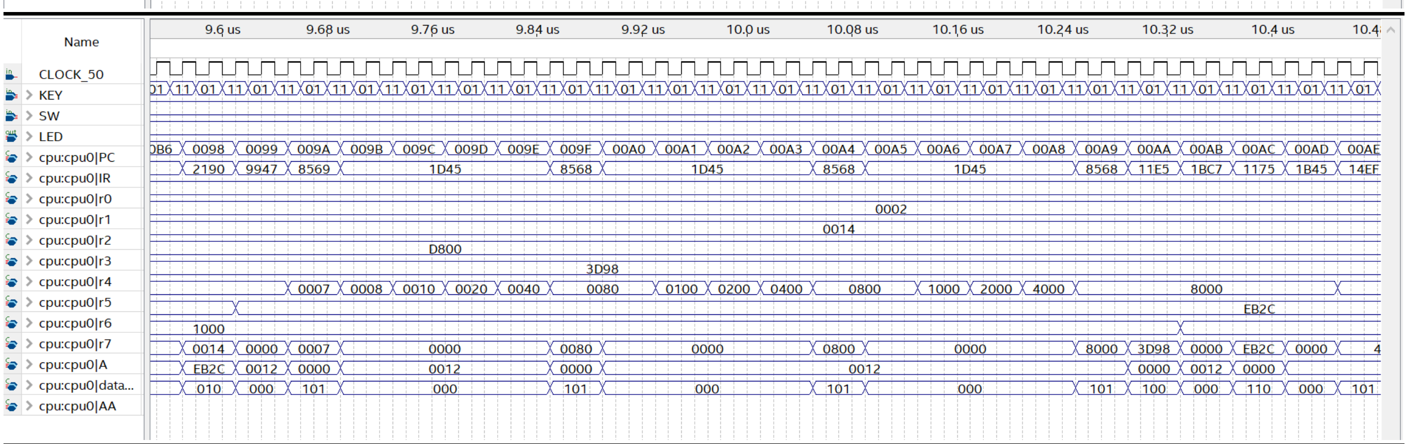

Name
5.2 us 5.28 us 5.36 us 5.44 us 5.52 us 5.6 us 5.68 us 5.76 us 5.84 us 5.92 us 6.0 us 6.08 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

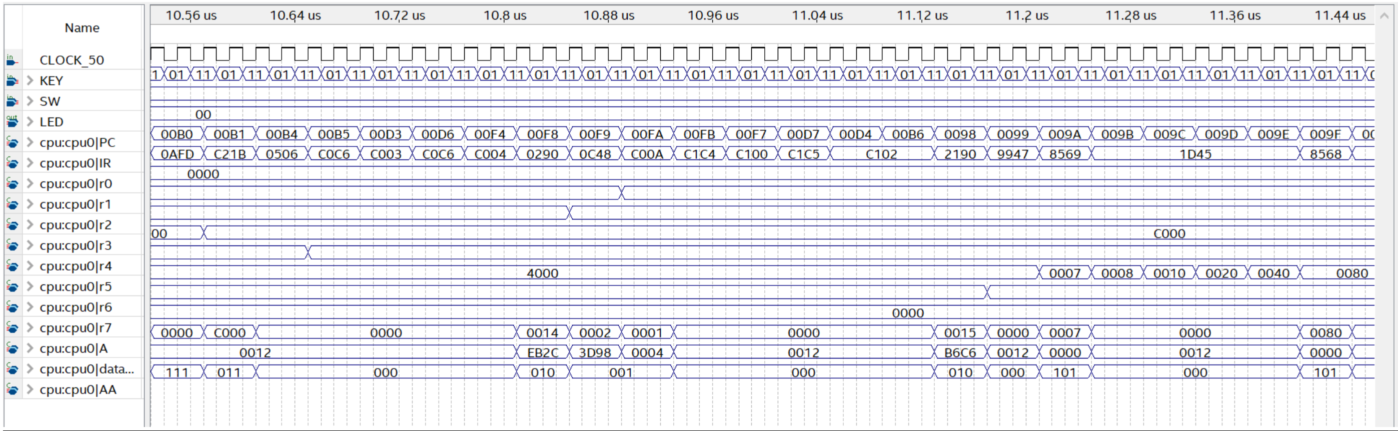

Name
6.08 us 6.16 us 6.24 us 6.32 us 6.4 us 6.48 us 6.56 us 6.64 us 6.72 us 6.8 us 6.88 us 6.96 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

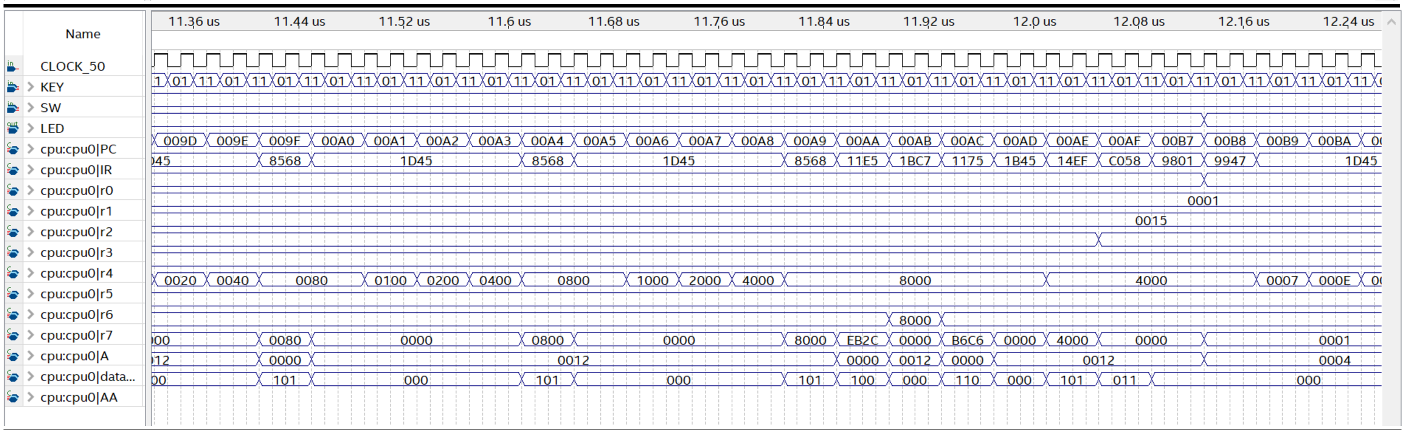

Name
6.96 us
7.04 us
7.12 us
7.2 us
7.28 us
7.36 us
7.44 us
7.52 us
7.6 us
7.68 us
7.76 us
7.84 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

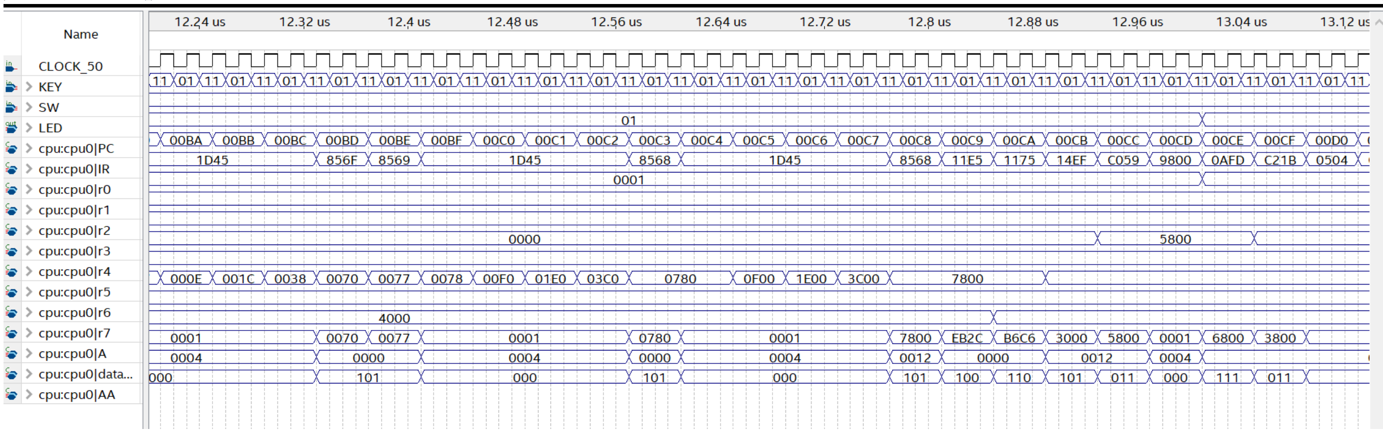

Name
7.84 us
7.92 us
8.0 us
8.08 us
8.16 us
8.24 us
8.32 us
8.4 us
8.48 us
8.56 us
8.64 us
8.72 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

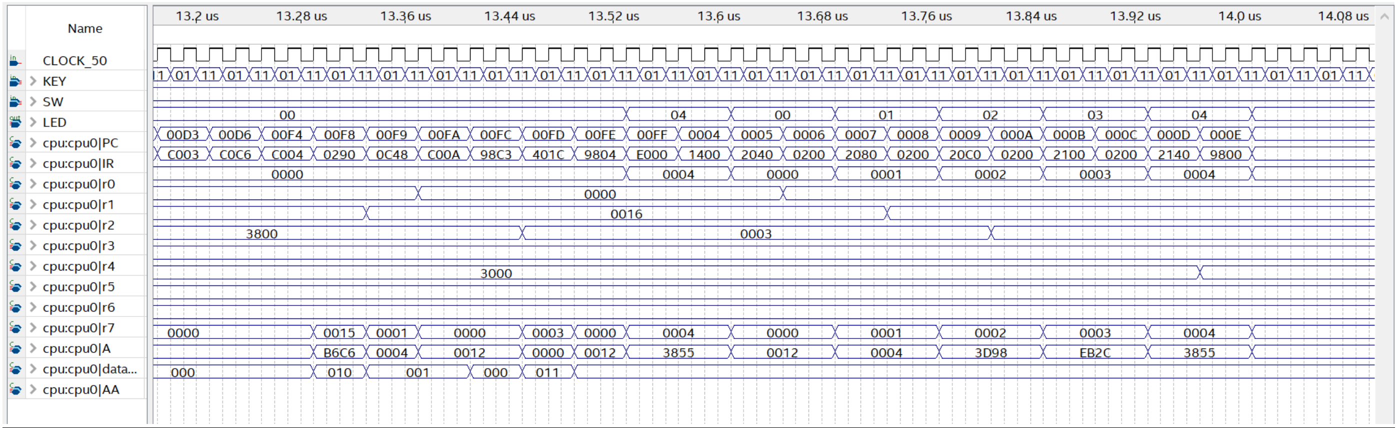

Name
8.72 us
8.8 us
8.88 us
8.96 us
9.04 us
9.12 us
9.2 us
9.28 us
9.36 us
9.44 us
9.52 us
9.6 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA

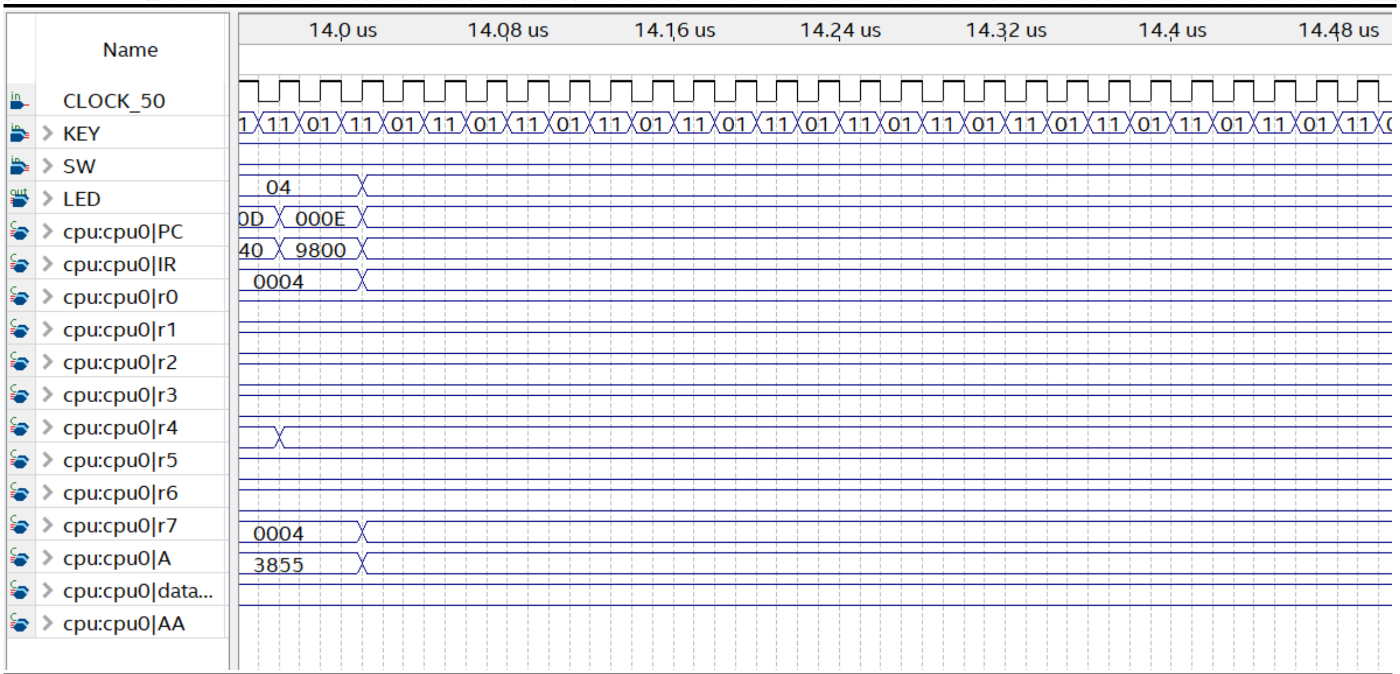

Name
9.6 us
9.68 us
9.76 us
9.84 us
9.92 us
10.0 us
10.08 us
10.16 us
10.24 us
10.32 us
10.4 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA


Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA
10.56 us
10.64 us
10.72 us
10.8 us
10.88 us
10.96 us
11.04 us
11.12 us
11.2 us
11.28 us
11.36 us
11.44 us


Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA
11.36 us
11.44 us
11.52 us
11.6 us
11.68 us
11.76 us
11.84 us
11.92 us
12.0 us
12.08 us
12.16 us
12.24 us


Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|data...
cpu:cpu0|AA
12.24 us
12.32 us
12.4 us
12.48 us
12.56 us
12.64 us
12.72 us
12.8 us
12.88 us
12.96 us
13.04 us
13.12 us





Test 2:

Floating Point values used in the test and their decimal values:

4AEC: 5.4609375

1DA0: 0.106445313

A75B: -0.239929199

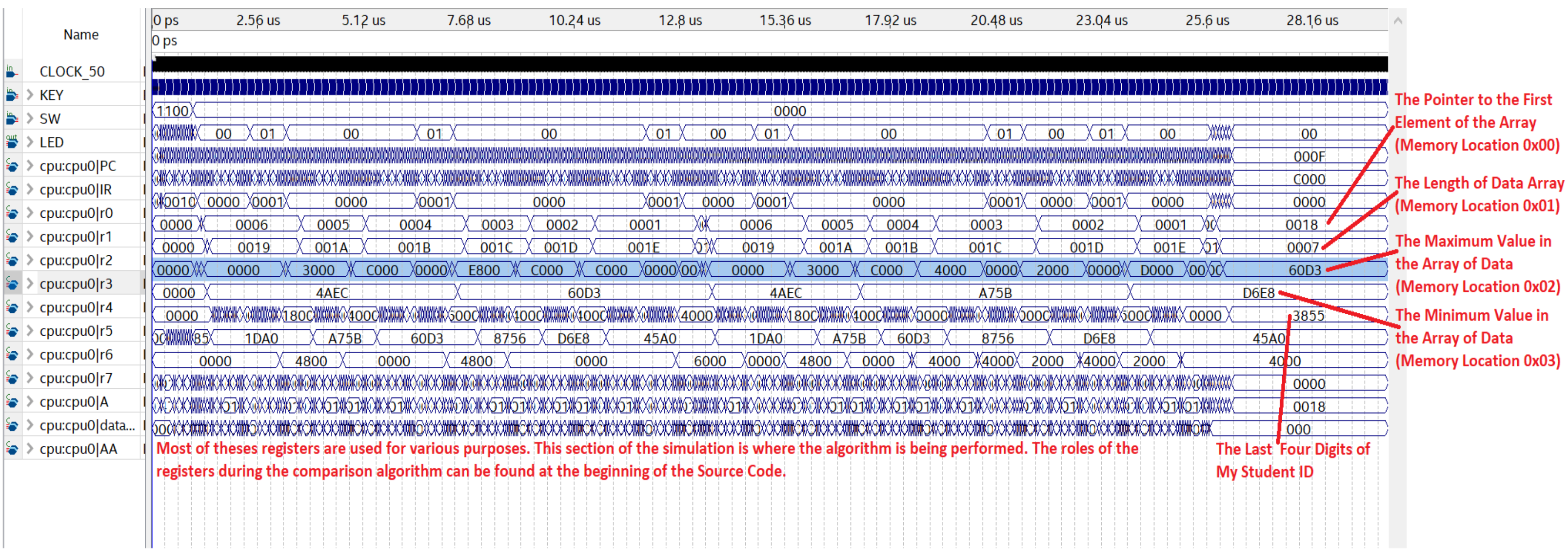
60D3: 35.296875 (This should be the Maximum Value)

8756: -0.014976501

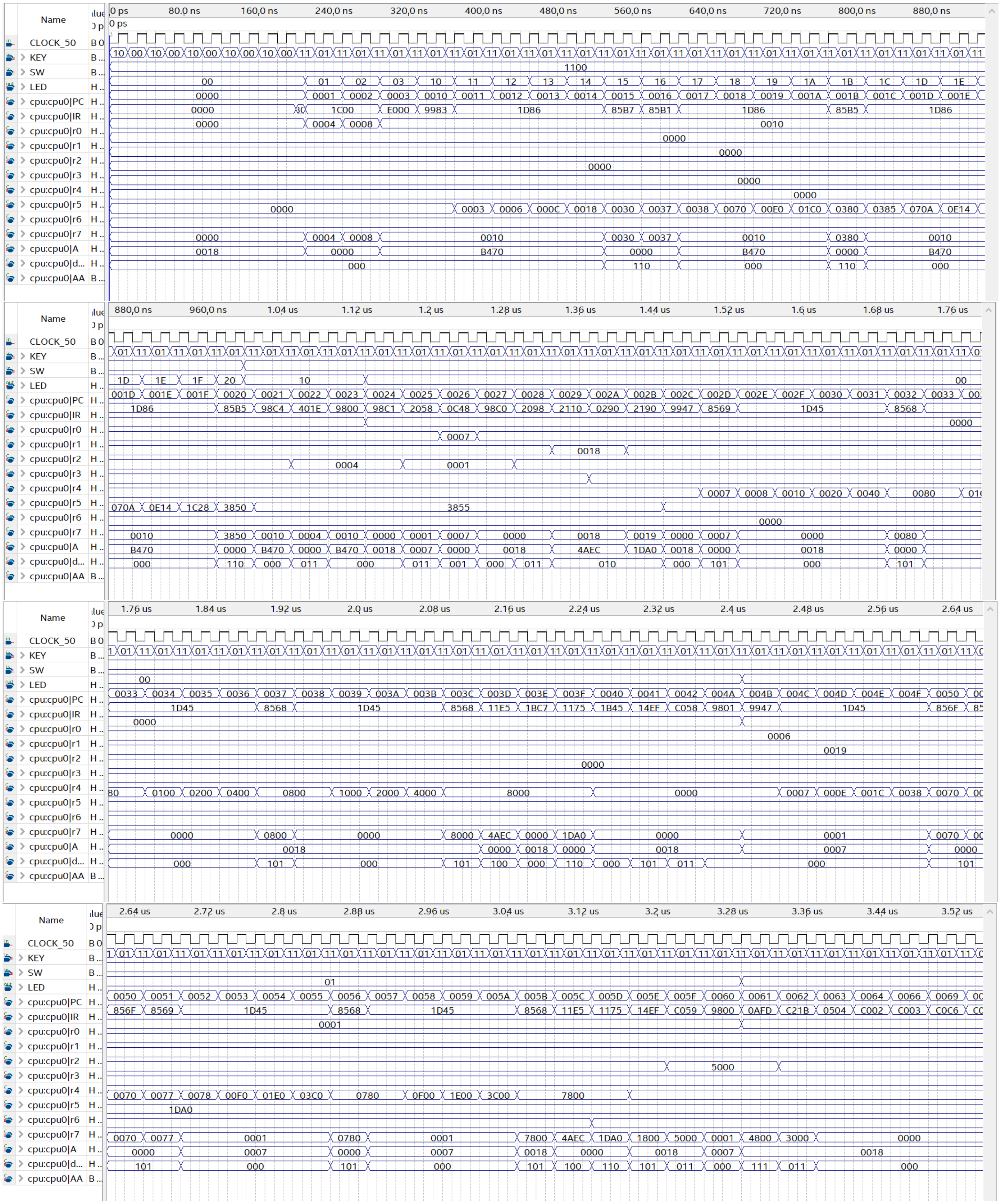
D6E8: -14.90625 (This should be the Minimum Value)

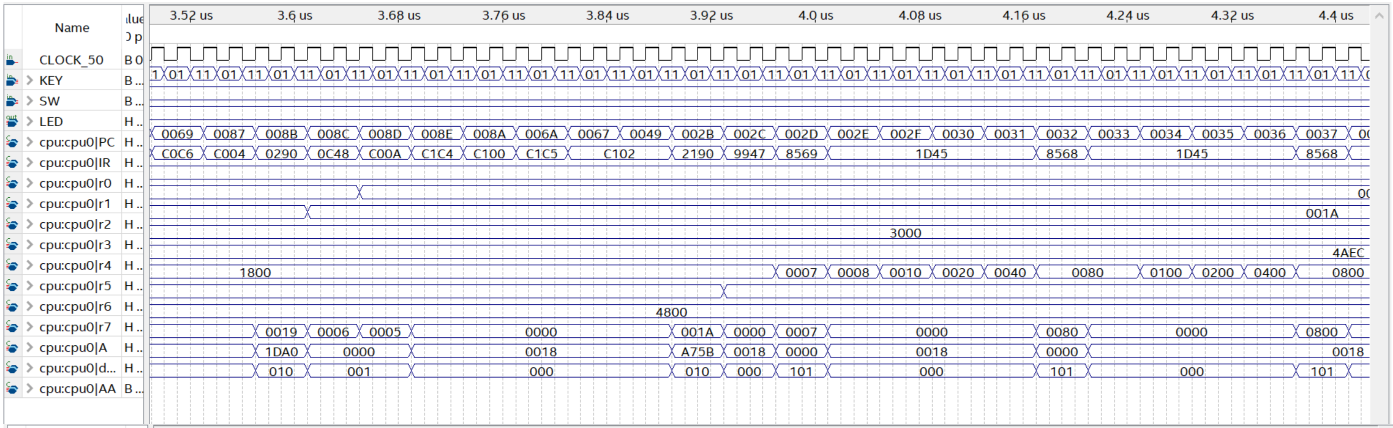
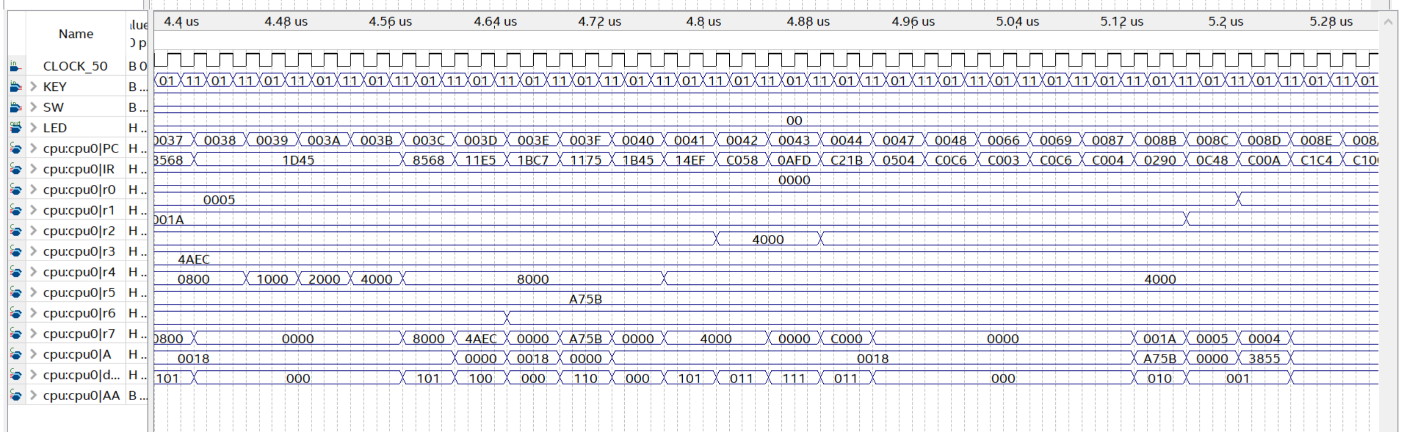
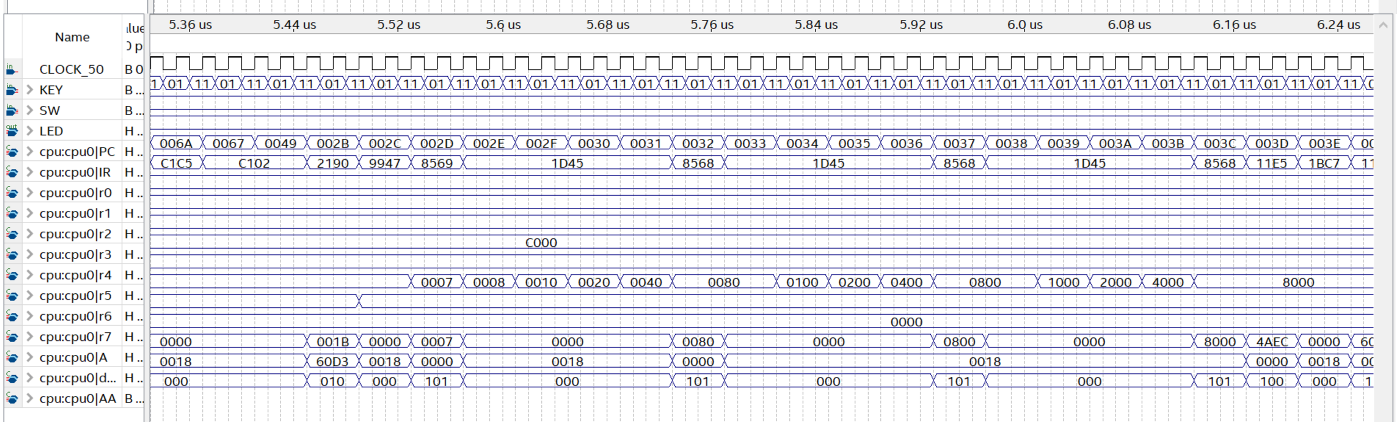
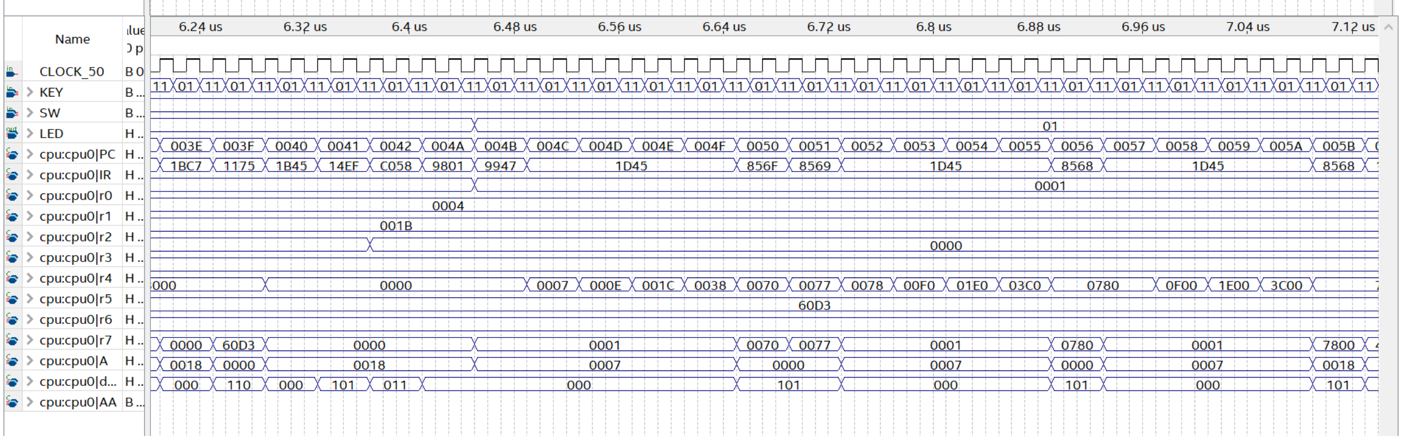
45A0: 3.40625

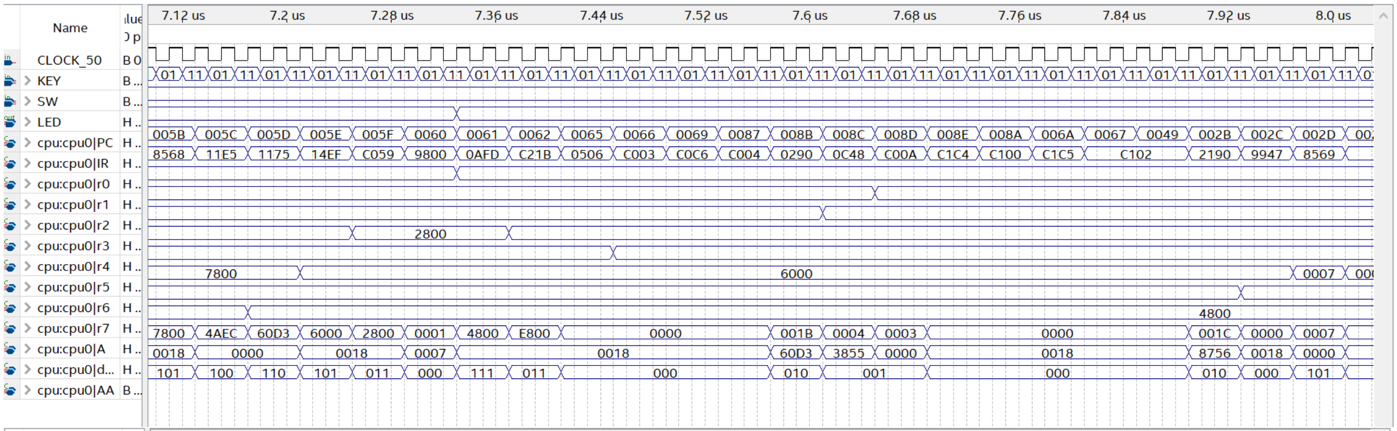
Full Simulation Results:

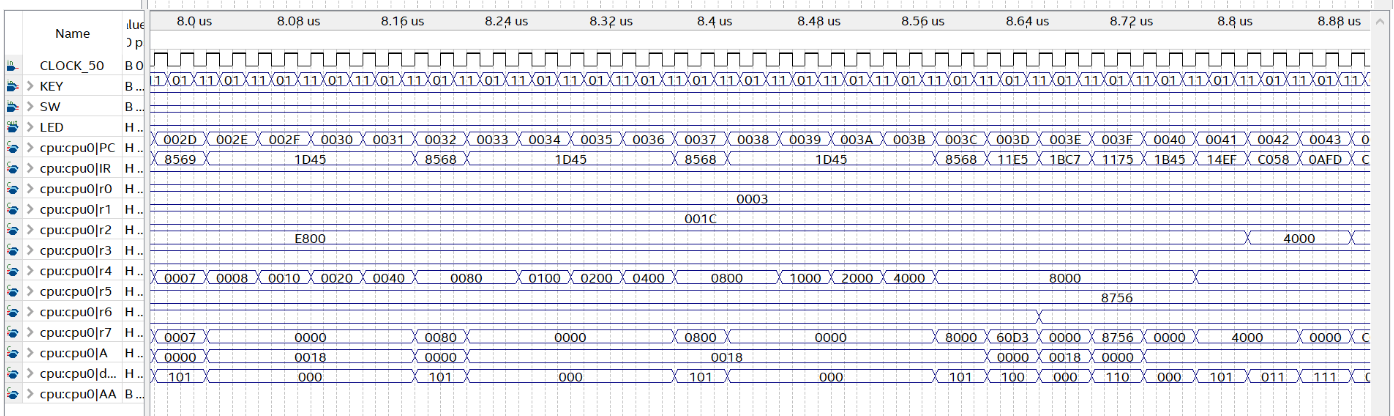


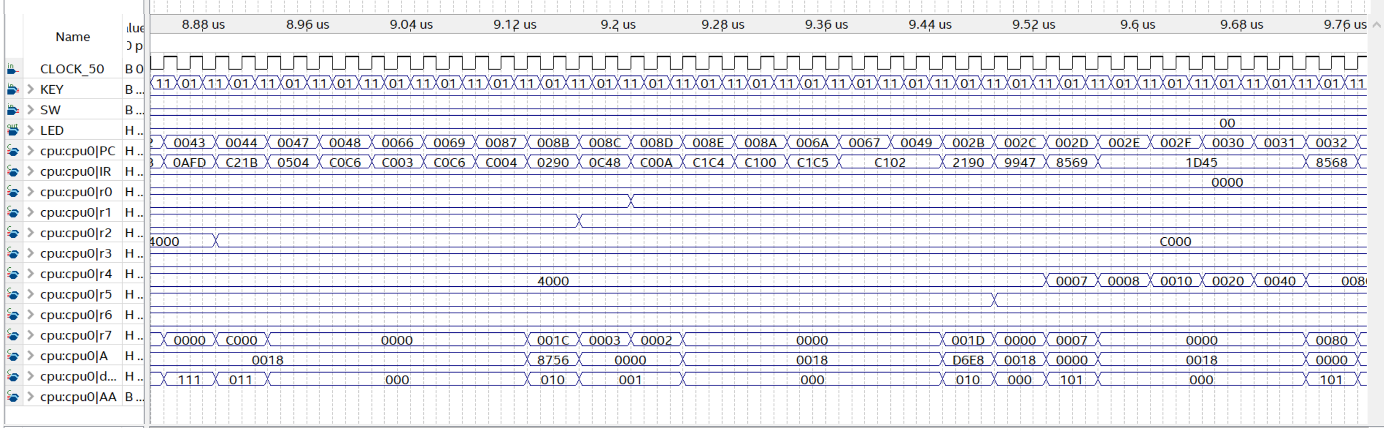
Test 2 Zoomed In Until Final Results of Simulation Are Generated:

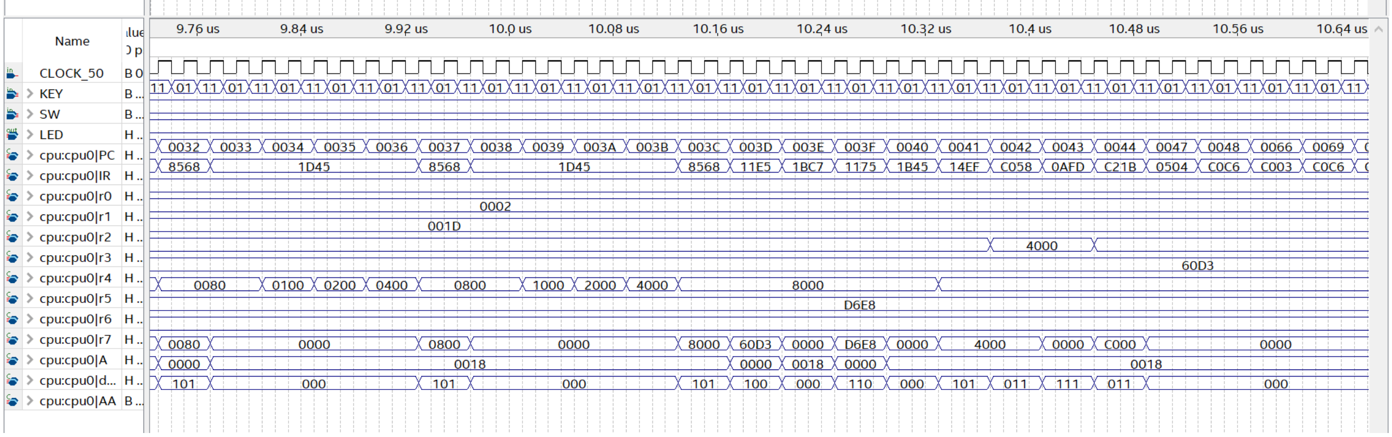


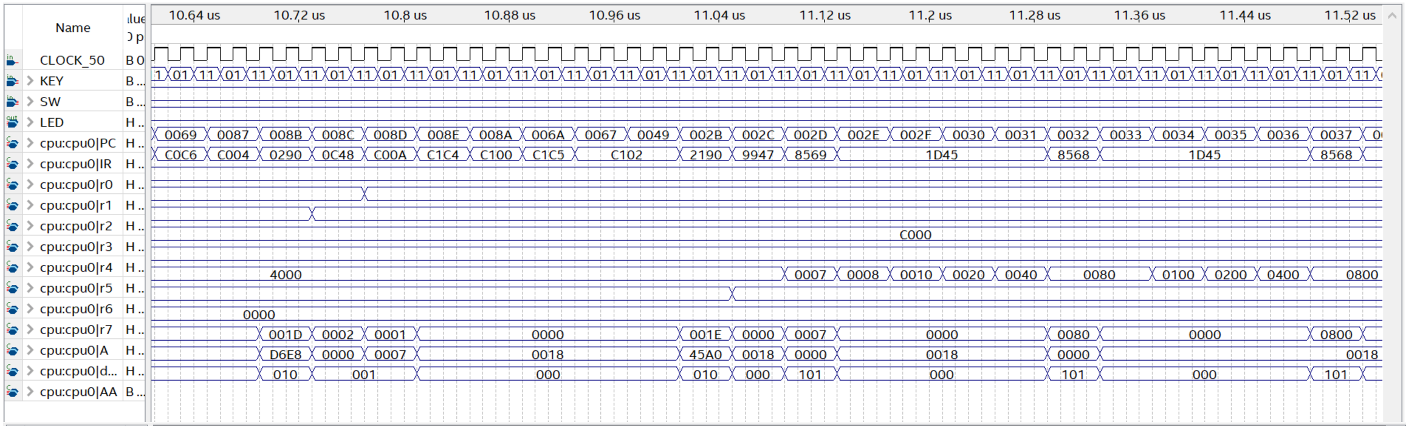
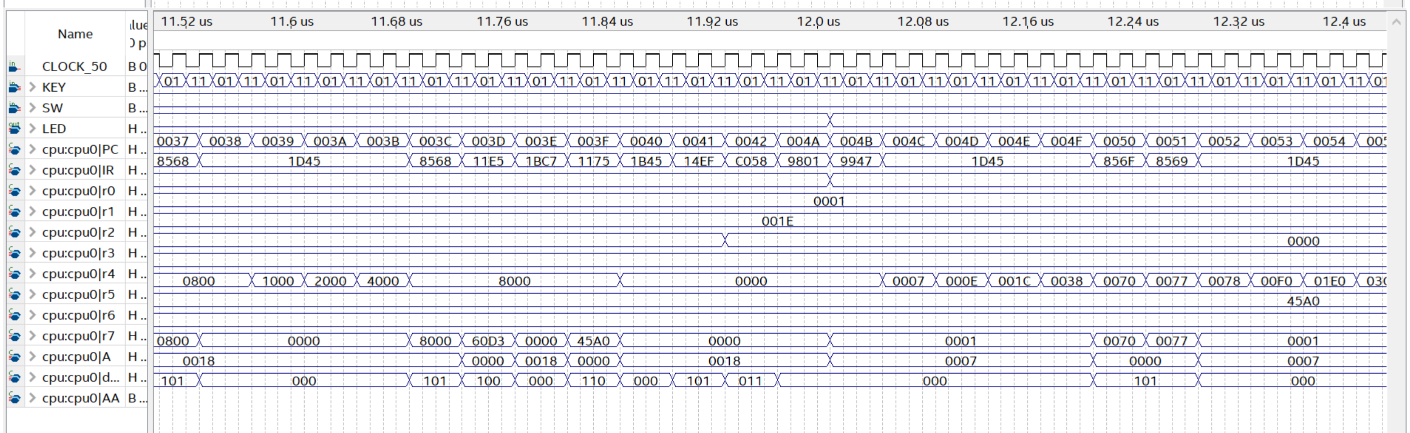
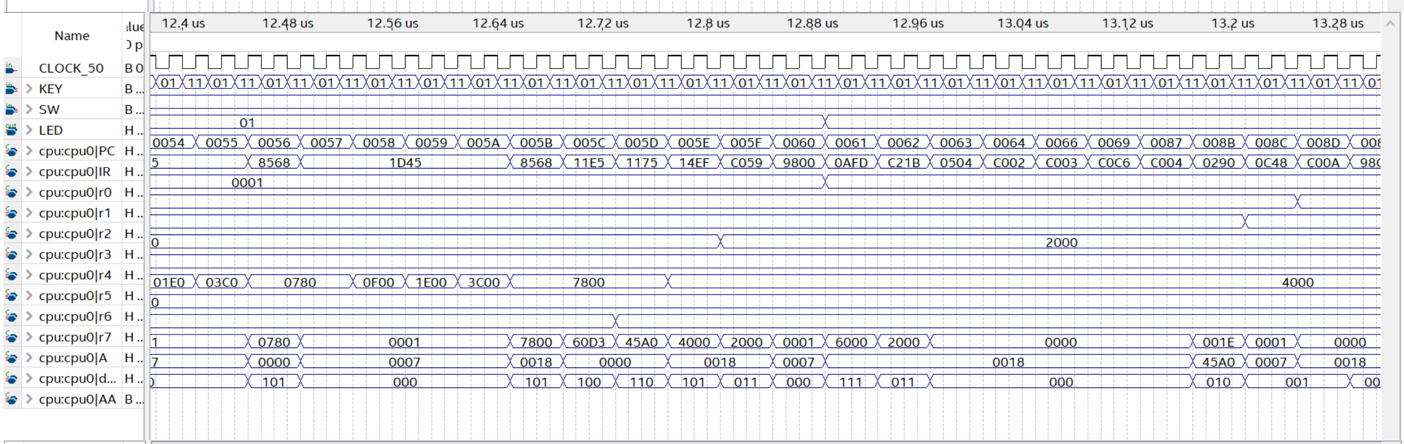
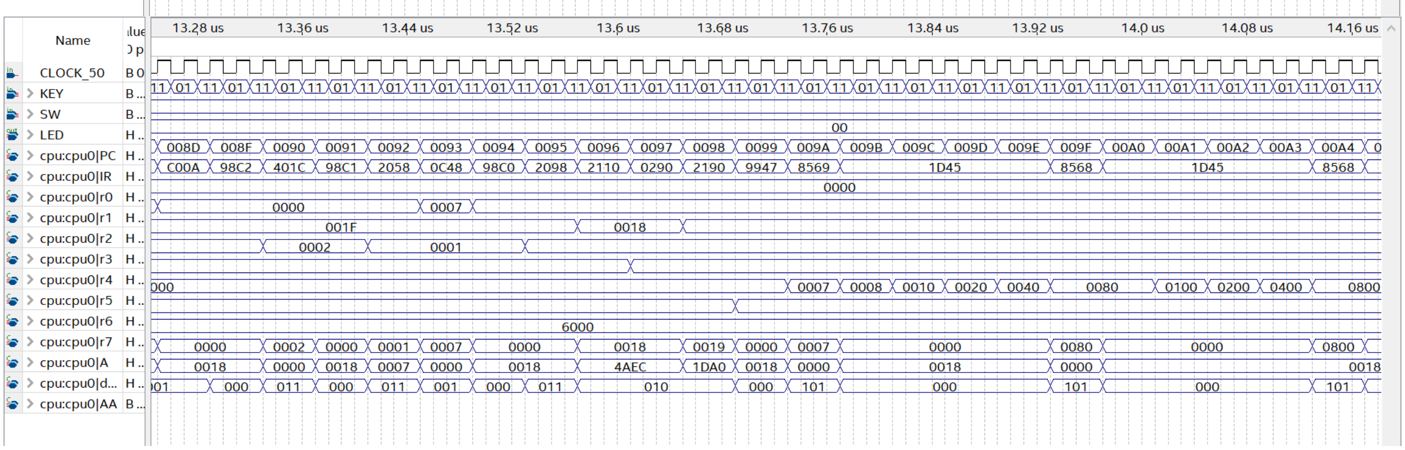

Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA
3.52 us 3.6 us 3.68 us 3.76 us 3.84 us 3.92 us 4.0 us 4.08 us 4.16 us 4.24 us 4.32 us 4.4 us
0069 0087 008B 008C 008D 008E 008A 006A 0067 0049 002B 002C 002D 002E 002F 0030 0031 0032 0033 0034 0035 0036 0037
C0C6 C004 0290 0C48 C00A C1C4 C100 C1C5 C102 2190 9947 8569 1D45 8568 1D45 8568
001A
3000
4AEC
1800 0007 0008 0010 0020 0040 0080 0100 0200 0400 0800
4800
0019 0006 0005 0000 001A 0000 0007 0000 0080 0000 0800
1DA0 0000 0018 A75B 0018 0000 0018 0000 0018
010 001 000 010 000 101 000 101 000 101

Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA
4.4 us 4.48 us 4.56 us 4.64 us 4.72 us 4.8 us 4.88 us 4.96 us 5.04 us 5.12 us 5.2 us 5.28 us
00
0037 0038 0039 003A 003B 003C 003D 003E 003F 0040 0041 0042 0043 0044 0047 0048 0066 0069 0087 008B 008C 008D 008E
8568 1D45 8568 11E5 1BC7 1175 1B45 14EF C058 0AFD C21B 0504 C0C6 C003 C0C6 C004 0290 0C48 C00A C1C4
0000
0005
001A
4000
4AEC
0800 1000 2000 4000 8000 4000
A75B
0800 0000 8000 4AEC 0000 A75B 0000 4000 0000 C000 0000 001A 0005 0004
0018 0000 0018 0000 0018 A75B 0000 3855
101 000 101 100 000 110 000 101 011 111 011 000 010 001

Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA
5.36 us 5.44 us 5.52 us 5.6 us 5.68 us 5.76 us 5.84 us 5.92 us 6.0 us 6.08 us 6.16 us 6.24 us
006A 0067 0049 002B 002C 002D 002E 002F 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 003A 003B 003C 003D 003E
C1C5 C102 2190 9947 8569 1D45 8568 1D45 8568 1D45 8568 11E5 1BC7
C000
0007 0008 0010 0020 0040 0080 0100 0200 0400 0800 1000 2000 4000 8000
0000
0000 001B 0000 0007 0000 0080 0000 0800 0000 8000 4AEC 0000
0018 60D3 0018 0000 0018 0000 0018 0000 0018
000 010 000 101 000 101 000 101 000 101 100 000

Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA
6.24 us 6.32 us 6.4 us 6.48 us 6.56 us 6.64 us 6.72 us 6.8 us 6.88 us 6.96 us 7.04 us 7.12 us
01
003E 003F 0040 0041 0042 004A 004B 004C 004D 004E 004F 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 005B
1BC7 1175 1B45 14EF C058 9801 9947 1D45 856F 8569 1D45 8568 1D45 8568
0001
0004
001B
0000
0000 0007 000E 001C 0038 0070 0077 0078 00F0 01E0 03C0 0780 0F00 1E00 3C00
60D3
0000 60D3 0000 0001 0070 0077 0001 0780 0001 7800
0018 0000 0018 0007 0000 0007 0000 0007 0018
000 110 000 101 011 000 101 000 101 000 101

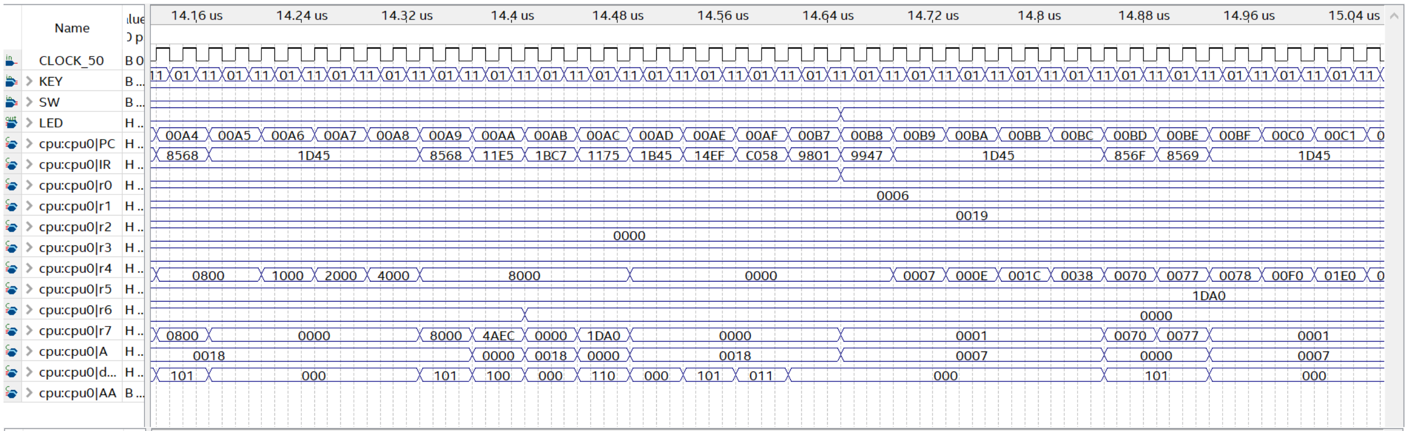

Name
7.12 us
7.2 us
7.28 us
7.36 us
7.44 us
7.52 us
7.6 us
7.68 us
7.76 us
7.84 us
7.92 us
8.0 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

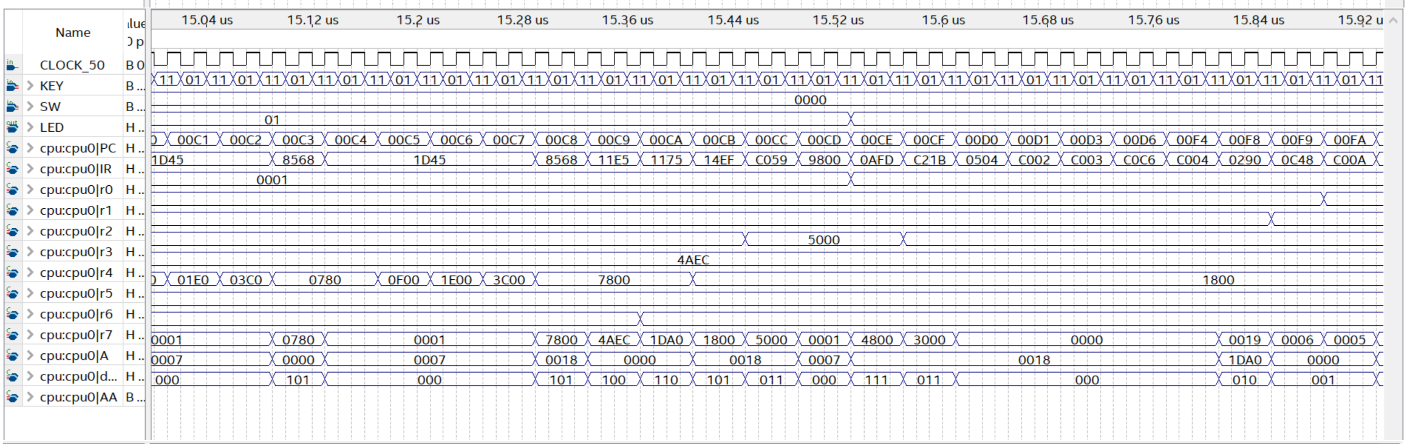

Name
8.0 us
8.08 us
8.16 us
8.24 us
8.32 us
8.4 us
8.48 us
8.56 us
8.64 us
8.72 us
8.8 us
8.88 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

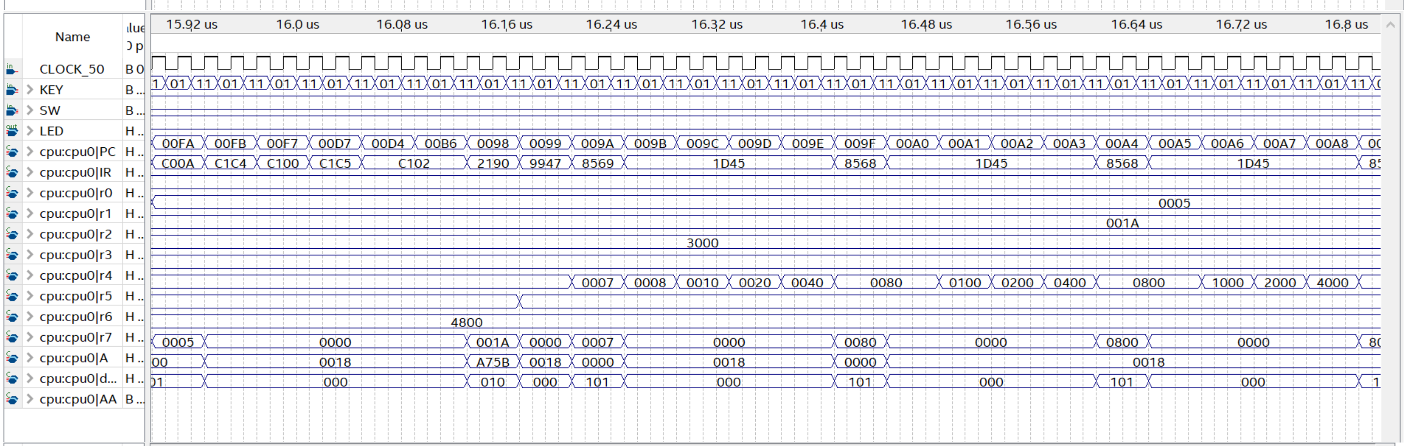

Name
8.88 us
8.96 us
9.04 us
9.12 us
9.2 us
9.28 us
9.36 us
9.44 us
9.52 us
9.6 us
9.68 us
9.76 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

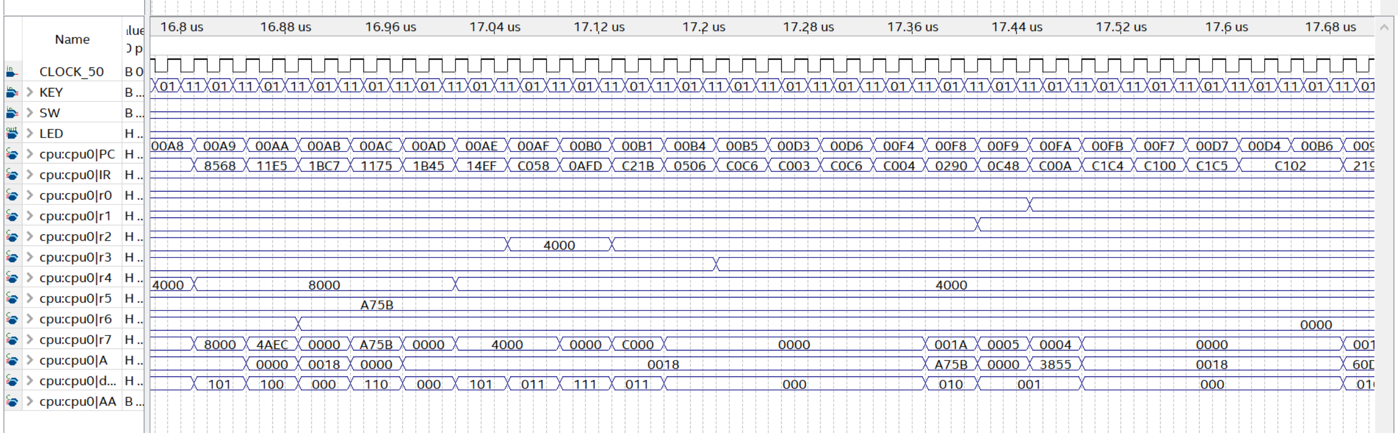

Name
9.76 us
9.84 us
9.92 us
10.0 us
10.08 us
10.16 us
10.24 us
10.32 us
10.4 us
10.48 us
10.56 us
10.64 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

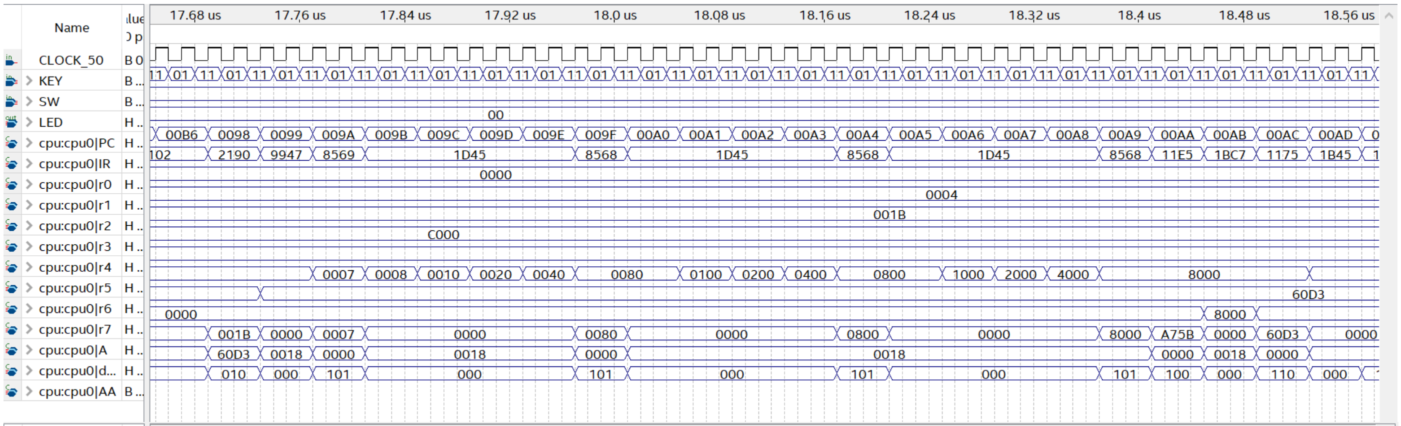

10.64 us 10.72 us 10.8 us 10.88 us 10.96 us 11.04 us 11.12 us 11.2 us 11.28 us 11.36 us 11.44 us 11.52 us
Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

11.52 us 11.6 us 11.68 us 11.76 us 11.84 us 11.92 us 12.0 us 12.08 us 12.16 us 12.24 us 12.32 us 12.4 us
Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

12.4 us 12.48 us 12.56 us 12.64 us 12.72 us 12.8 us 12.88 us 12.96 us 13.04 us 13.12 us 13.2 us 13.28 us
Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

13.28 us 13.36 us 13.44 us 13.52 us 13.6 us 13.68 us 13.76 us 13.84 us 13.92 us 14.0 us 14.08 us 14.16 us
Name
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

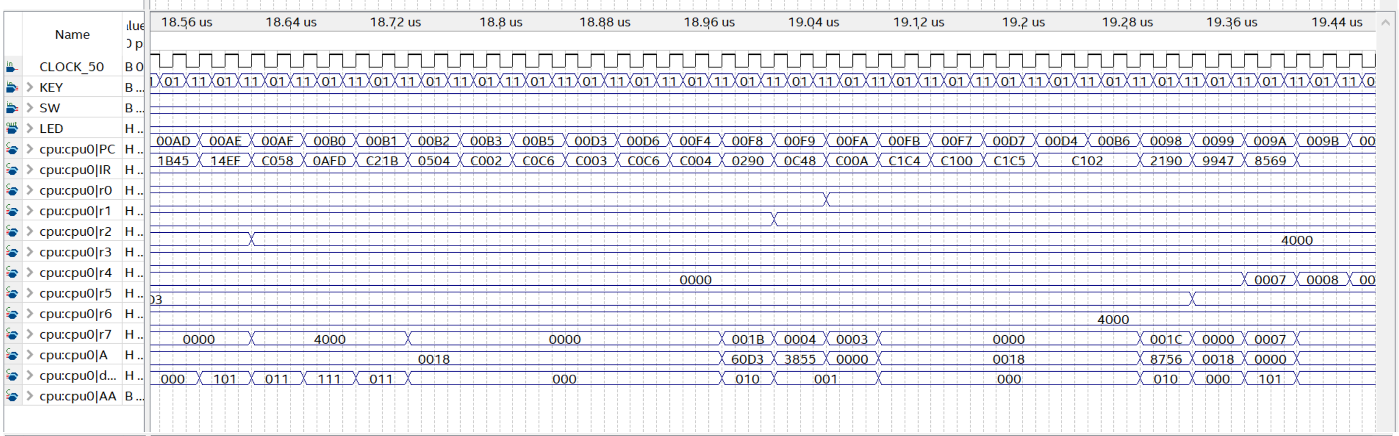

Name
14.16 us 14.24 us 14.32 us 14.4 us 14.48 us 14.56 us 14.64 us 14.72 us 14.8 us 14.88 us 14.96 us 15.04 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

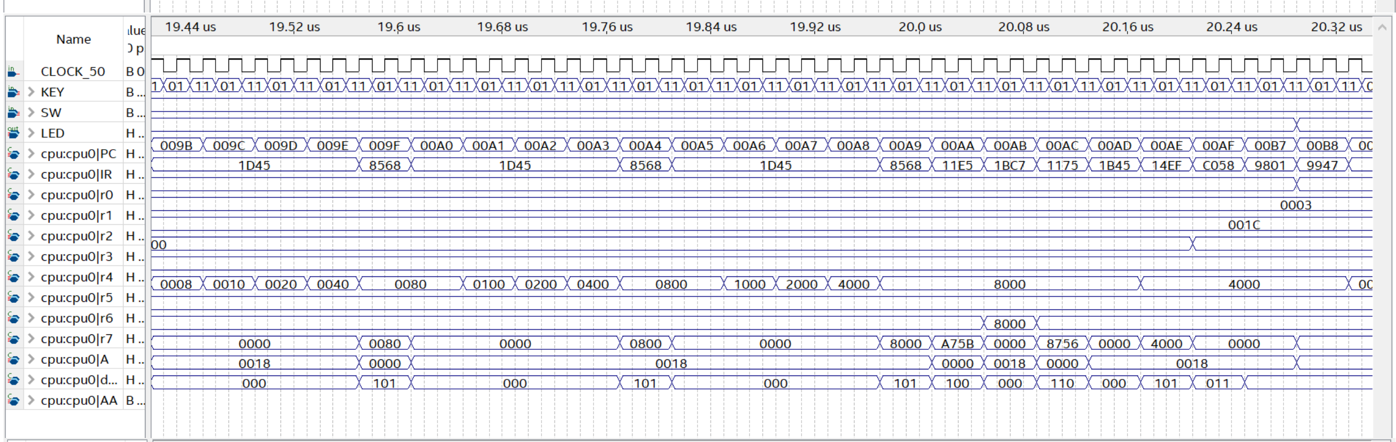

Name
15.04 us 15.12 us 15.2 us 15.28 us 15.36 us 15.44 us 15.52 us 15.6 us 15.68 us 15.76 us 15.84 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

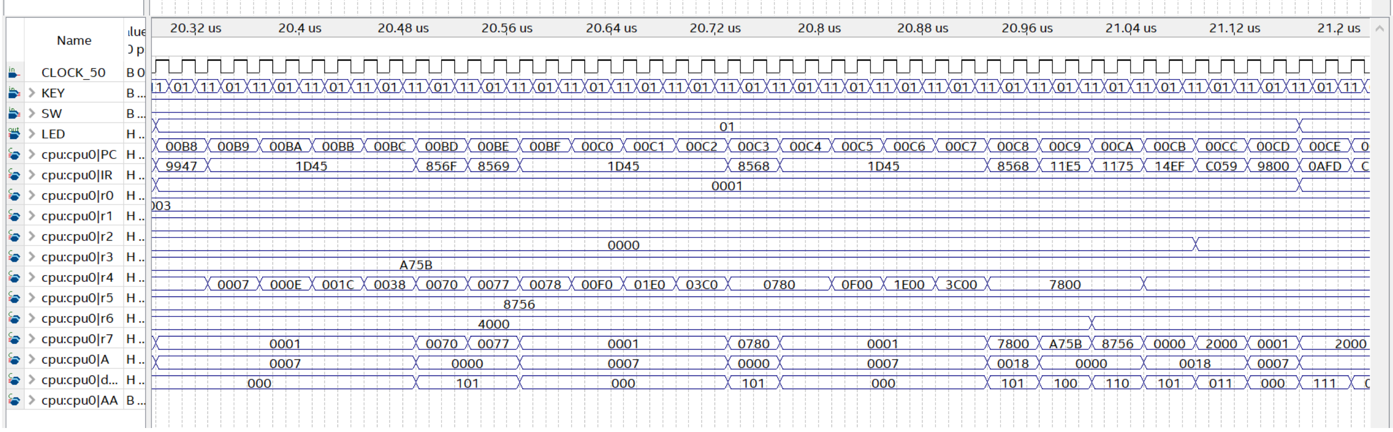

Name
15.92 us 16.0 us 16.08 us 16.16 us 16.24 us 16.32 us 16.4 us 16.48 us 16.56 us 16.64 us 16.72 us 16.8 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

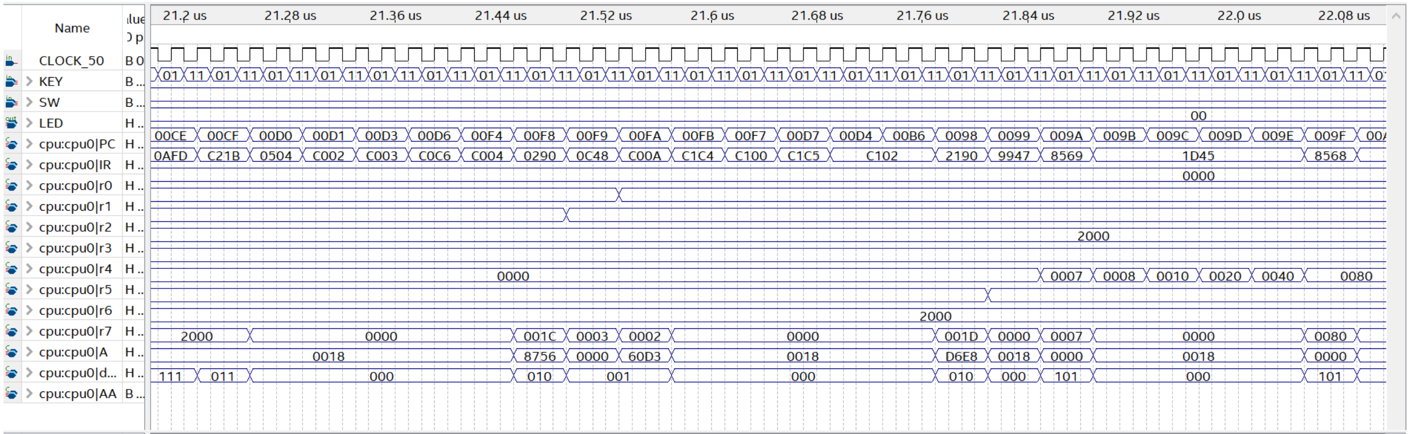

Name
16.8 us 16.88 us 16.96 us 17.04 us 17.12 us 17.2 us 17.28 us 17.36 us 17.44 us 17.52 us 17.6 us 17.68 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

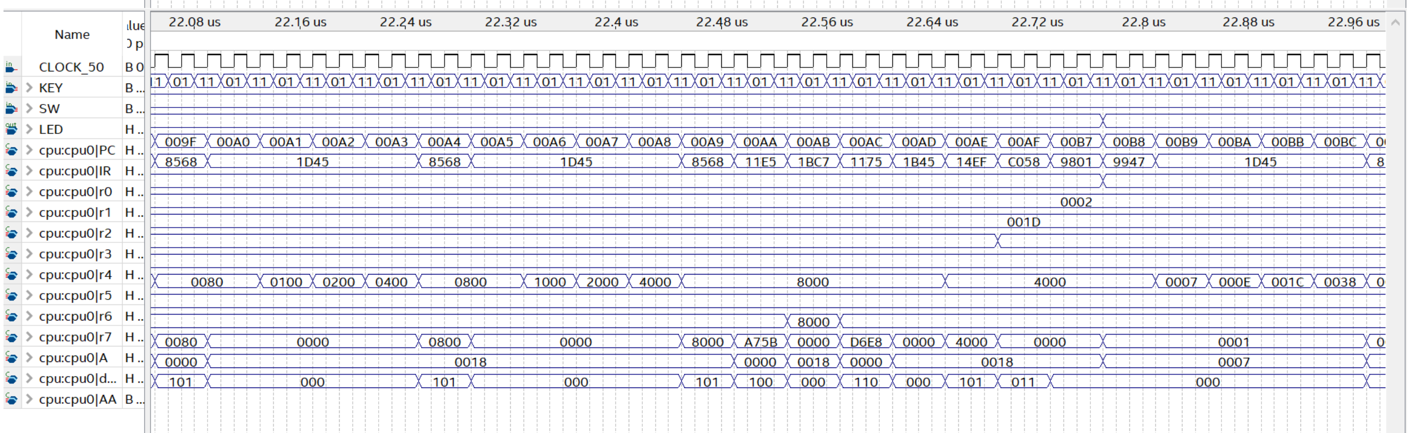

Name
17.68 us
17.76 us
17.84 us
17.92 us
18.0 us
18.08 us
18.16 us
18.24 us
18.32 us
18.4 us
18.48 us
18.56 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

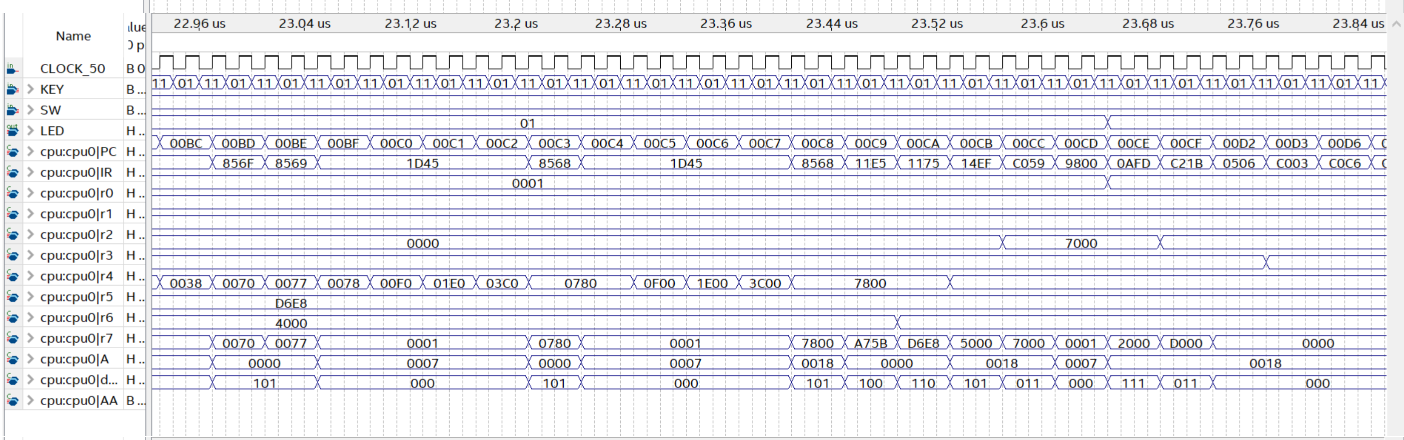

Name
18.56 us
18.64 us
18.72 us
18.8 us
18.88 us
18.96 us
19.04 us
19.12 us
19.2 us
19.28 us
19.36 us
19.44 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

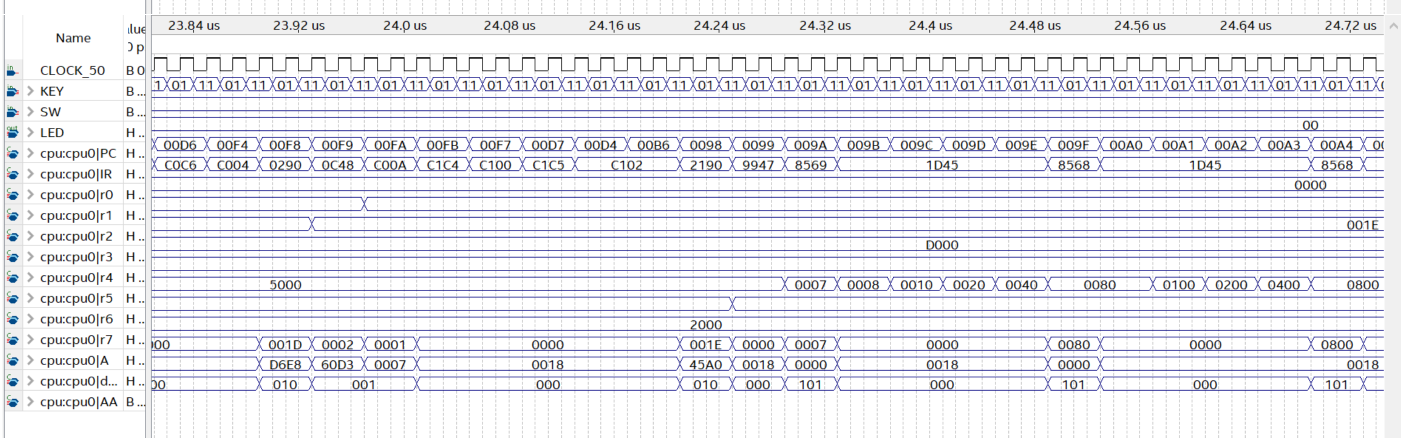

Name
19.44 us
19.52 us
19.6 us
19.68 us
19.76 us
19.84 us
19.92 us
20.0 us
20.08 us
20.16 us
20.24 us
20.32 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

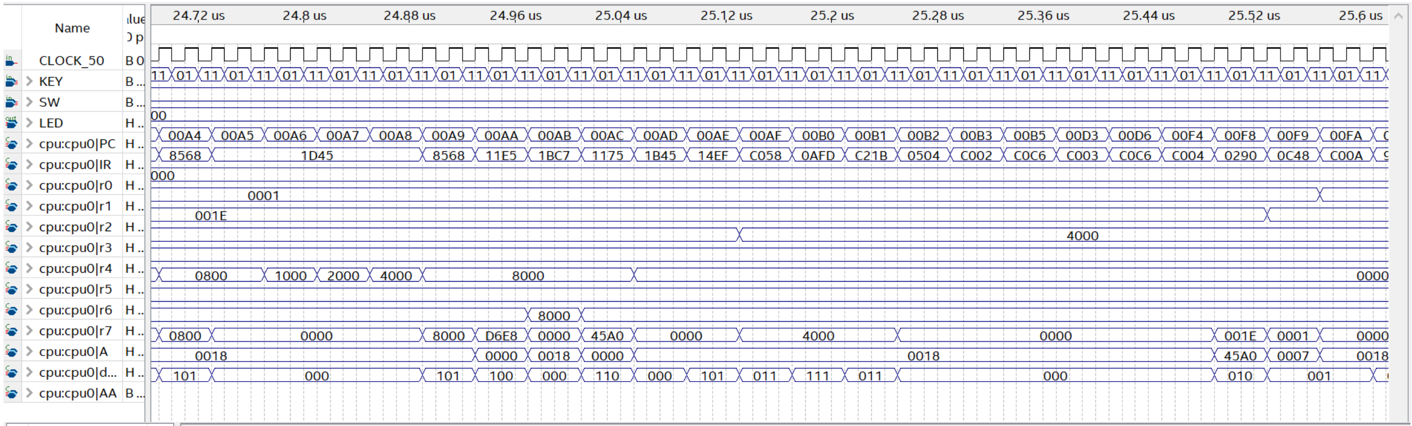

Name
20.32 us
20.4 us
20.48 us
20.56 us
20.64 us
20.72 us
20.8 us
20.88 us
20.96 us
21.04 us
21.12 us
21.2 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA

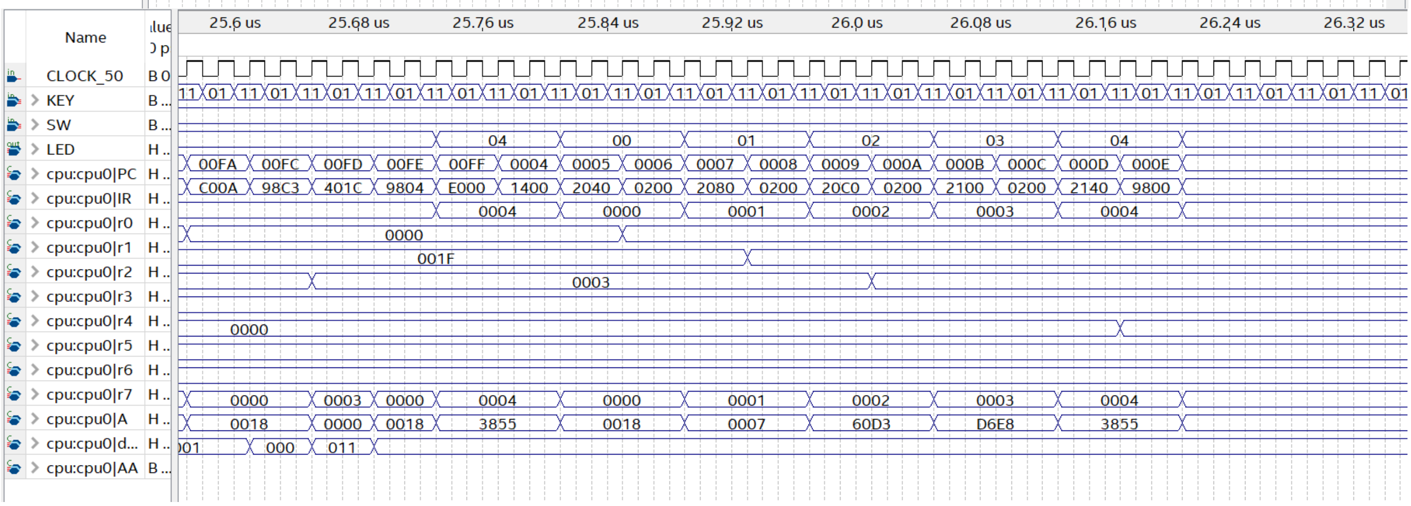

Name
21.2 us
21.28 us
21.36 us
21.44 us
21.52 us
21.6 us
21.68 us
21.76 us
21.84 us
21.92 us
22.0 us
22.08 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA


Name
22.08 us
22.16 us
22.24 us
22.32 us
22.4 us
22.48 us
22.56 us
22.64 us
22.72 us
22.8 us
22.88 us
22.96 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA


Name
22.96 us
23.04 us
23.12 us
23.2 us
23.28 us
23.36 us
23.44 us
23.52 us
23.6 us
23.68 us
23.76 us
23.84 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA


Name
23.84 us
23.92 us
24.0 us
24.08 us
24.16 us
24.24 us
24.32 us
24.4 us
24.48 us
24.56 us
24.64 us
24.72 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA


Name
24.72 us
24.8 us
24.88 us
24.96 us
25.04 us
25.12 us
25.2 us
25.28 us
25.36 us
25.44 us
25.52 us
25.6 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA


Name
25.6 us
25.68 us
25.76 us
25.84 us
25.92 us
26.0 us
26.08 us
26.16 us
26.24 us
26.32 us
CLOCK_50
KEY
SW
LED
cpu:cpu0|PC
cpu:cpu0|IR
cpu:cpu0|r0
cpu:cpu0|r1
cpu:cpu0|r2
cpu:cpu0|r3
cpu:cpu0|r4
cpu:cpu0|r5
cpu:cpu0|r6
cpu:cpu0|r7
cpu:cpu0|A
cpu:cpu0|d...
cpu:cpu0|AA