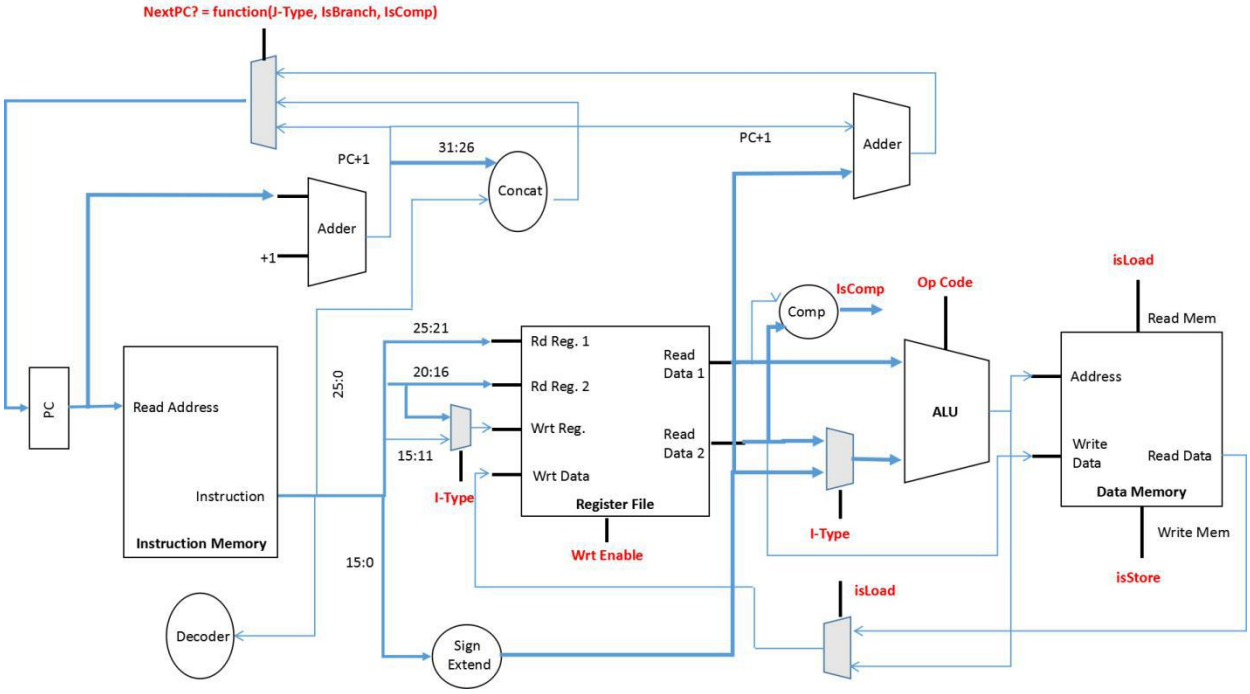
EL-GY 6463 Final Project Report

NYU-6463 Processor Design

And RC5 Implementations

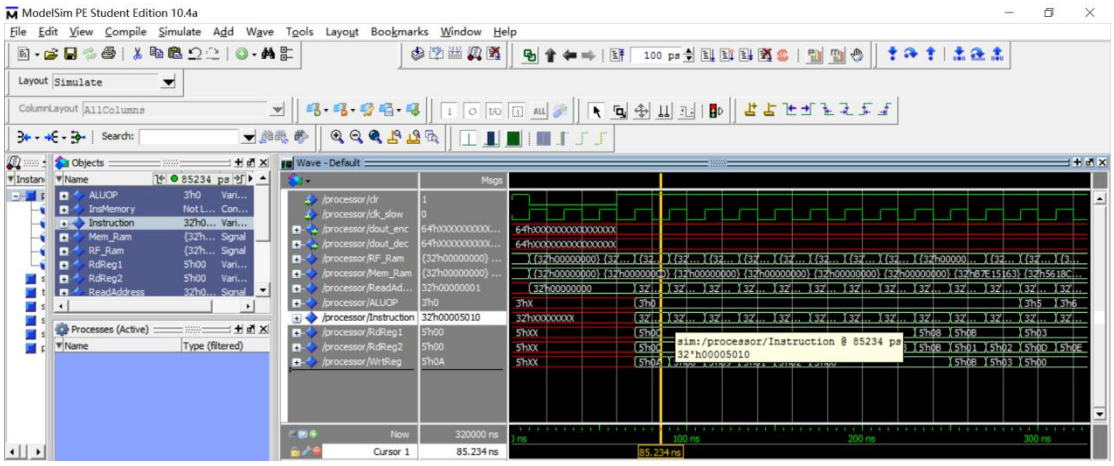
Minghan Jiang	mj1846
Kaixiang Ye	ky934
Kang Liu	kl1807
Zhen Mao	zm720
Yi Sun	ys2900

- **Block Diagram**



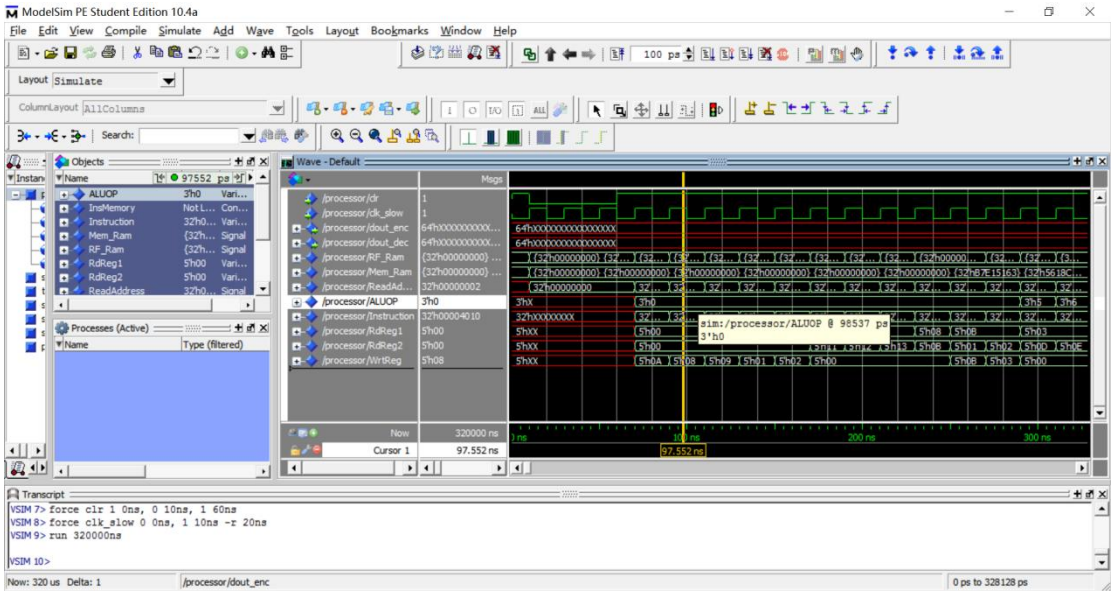
- **ALU/Decoder**

Case 1: ADD



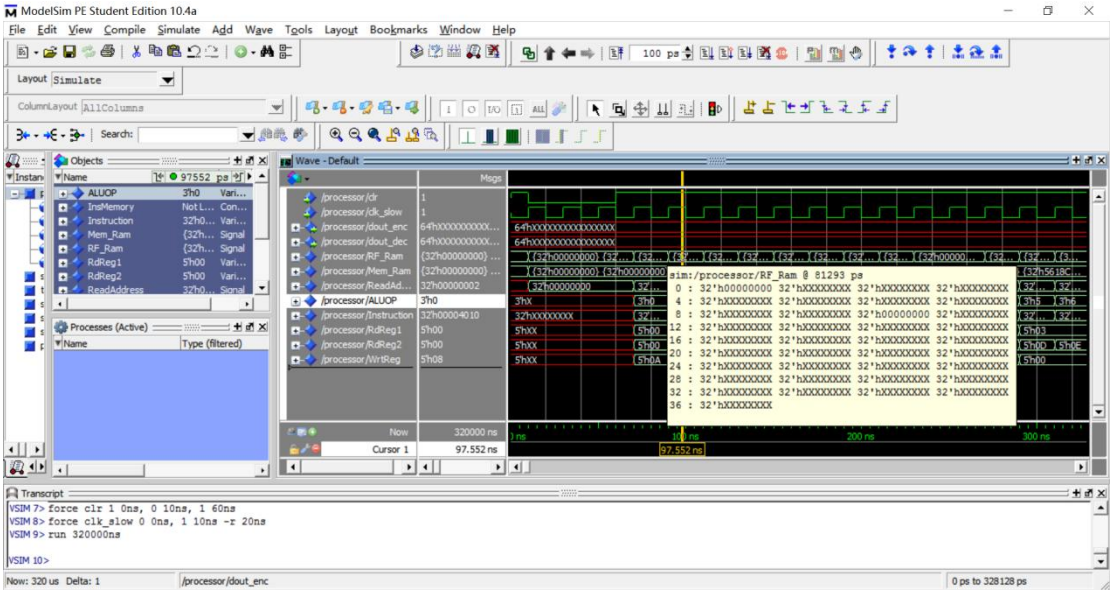
Instruction: ADD R0 R0 R10

Checkpoint: Decoder Unit



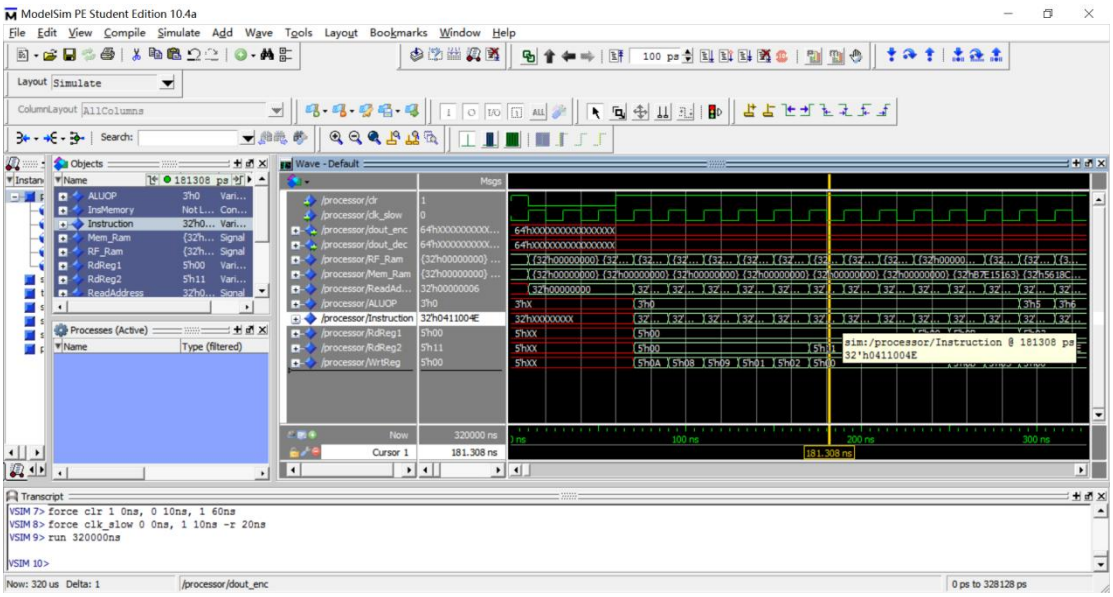
The Decoder successfully identifies this instruction as an R-type ADD and outputs Op code signal as ADD into ALU unit.

Checkpoint: ALU Unit



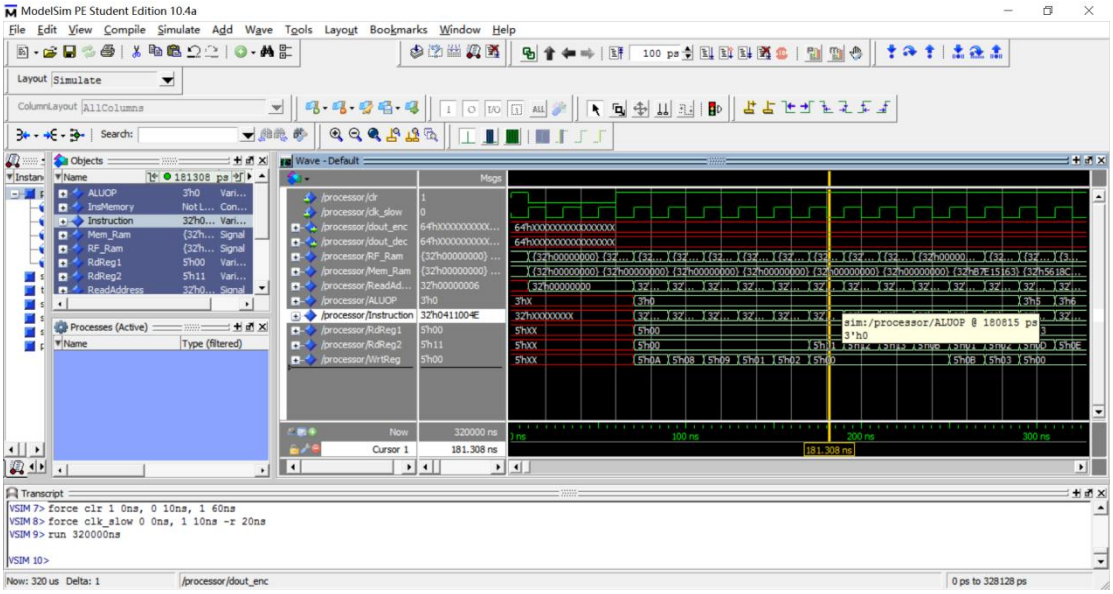
The ALU reads both input values from register R0 with constant value 32'b0, performs ADD operation to calculate the result and writes back the result 32'b0 into register R10.

Case 2: ADDI



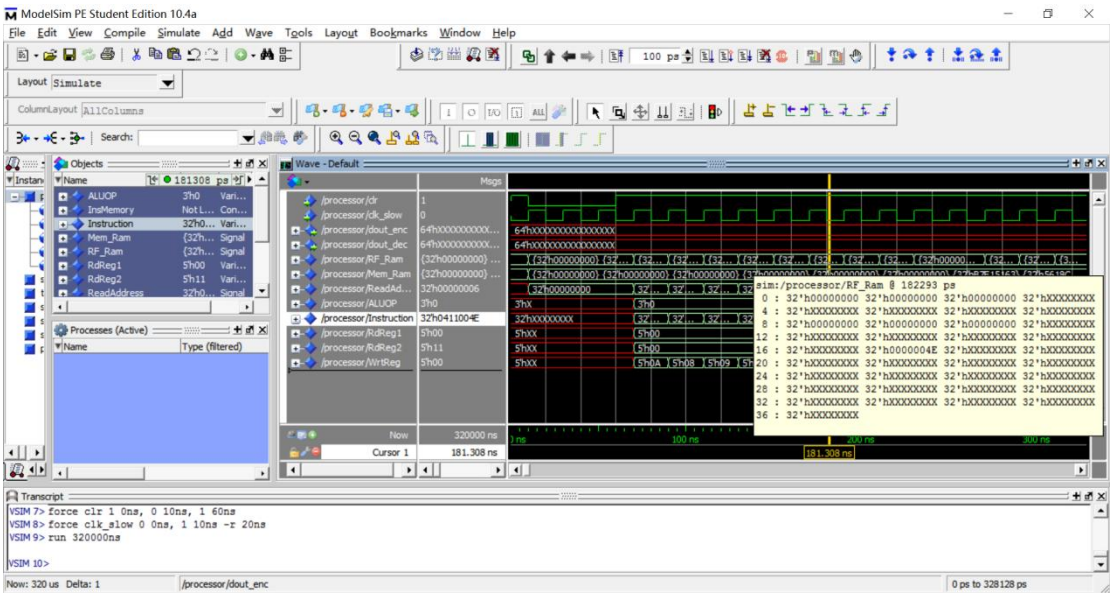
Instruction: ADDI R0 R17 78

Checkpoint: Decoder Unit



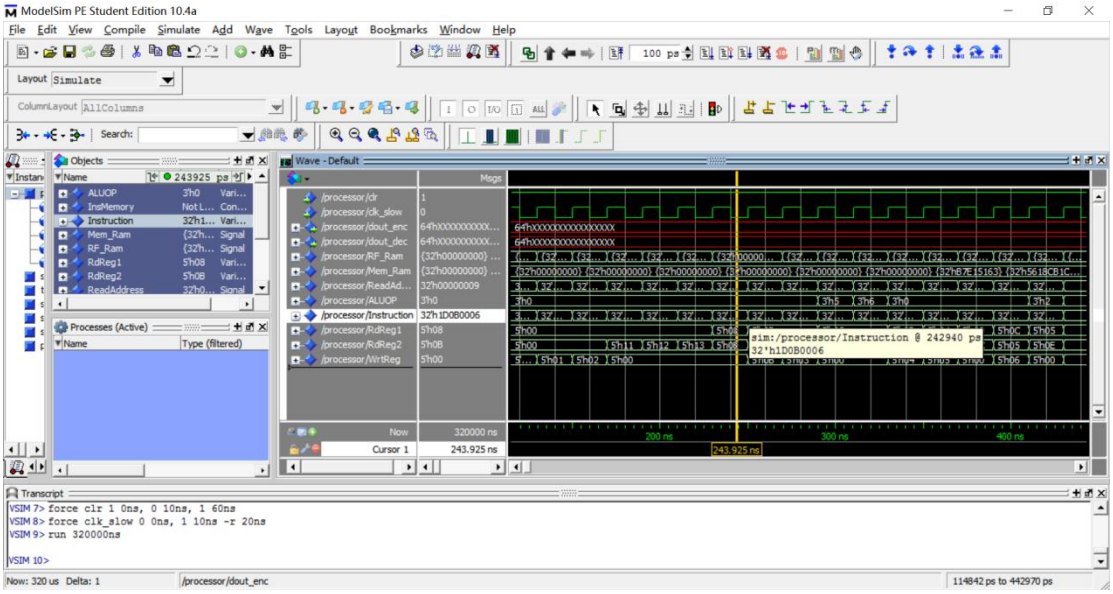
The Decoder successfully identifies this instruction as an I-type ADDI, and still remains the Op code signal as ADD and passes it into ALU unit.

Checkpoint: ALU Unit



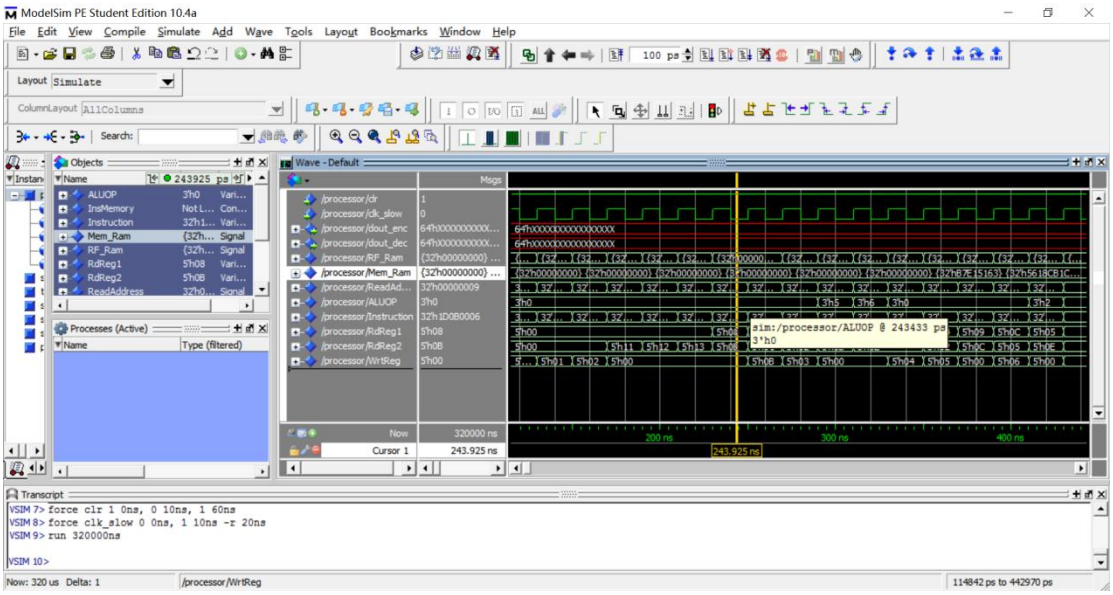
The ALU reads one input from register R0 with constant value 32’b0 and another immediate value 0x4E, performs ADD operation to calculate the result and writes back the result 0x4E into register R17.

Case 3: LW



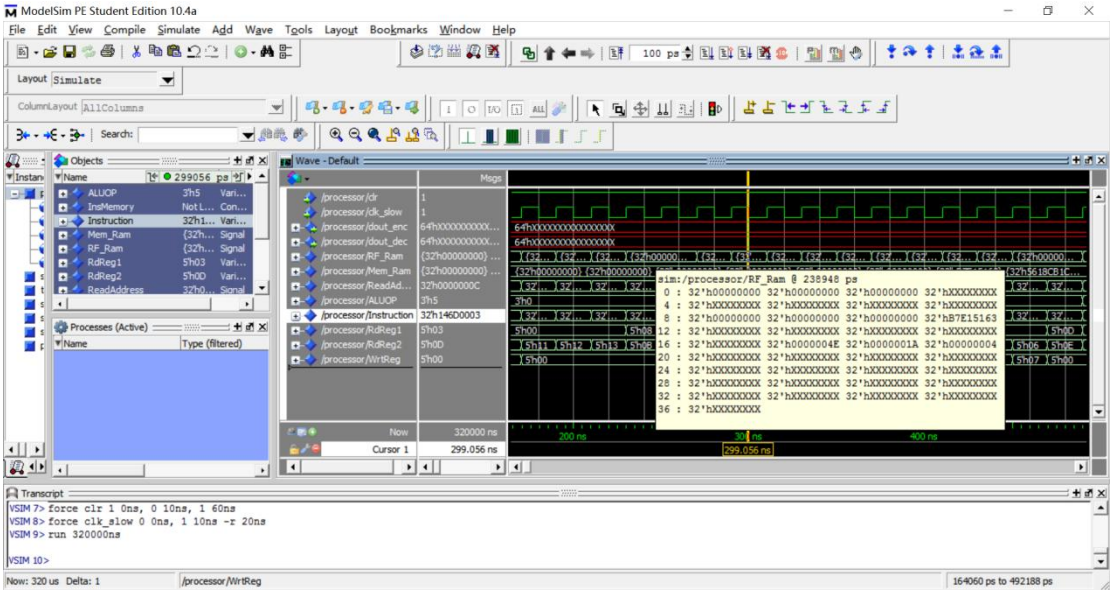
Instruction: LW R8 R11 S_ADD

Checkpoint: Decoder Unit



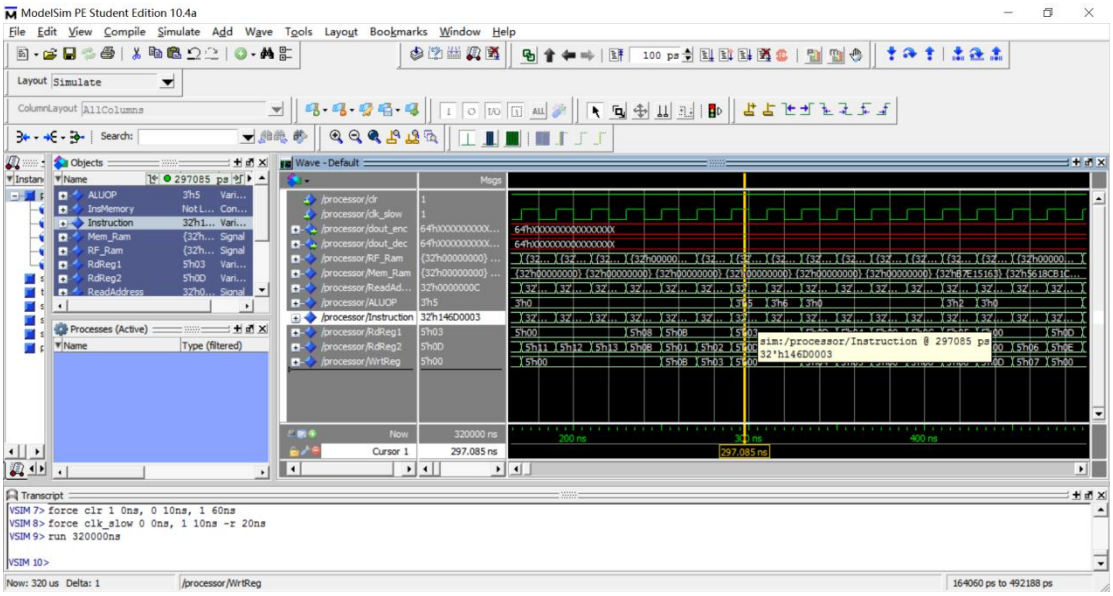
The Decoder successfully identifies this instruction as an I-type LW, and still remains the Op code signal as ADD and passes it into ALU unit.

Checkpoint: ALU Unit



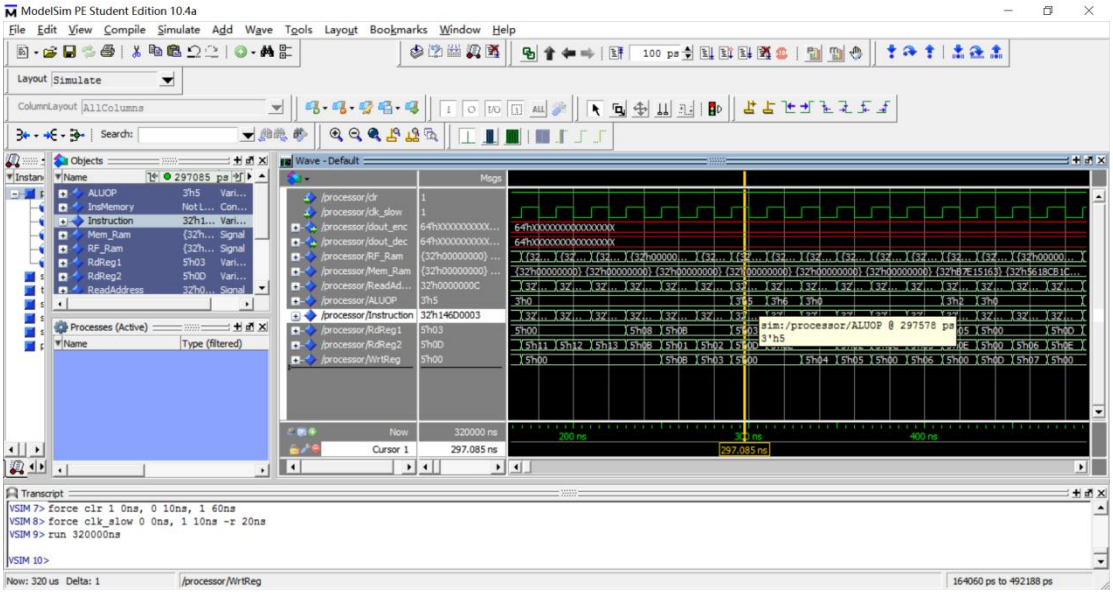
The ALU reads one input from register R8 with value 32'b0 and another immediate value 0x6, performs ADD operation to calculate the result and loads the value of address 0x6 in data memory into register R11.

Case 4: SHL



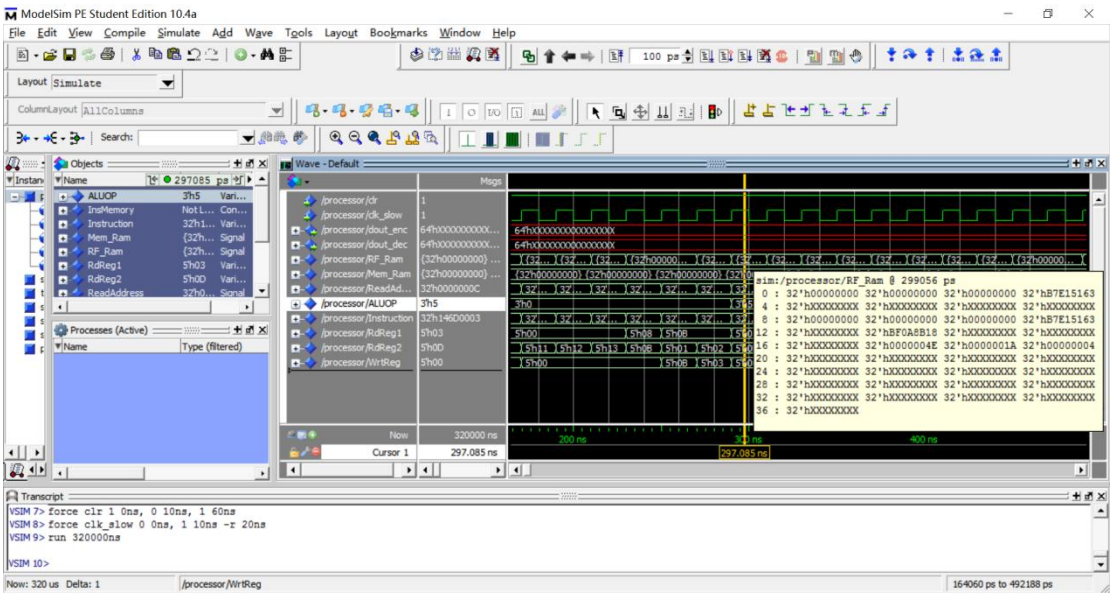
Instruction: SHL R3 R13 3

Checkpoint: Decoder Unit



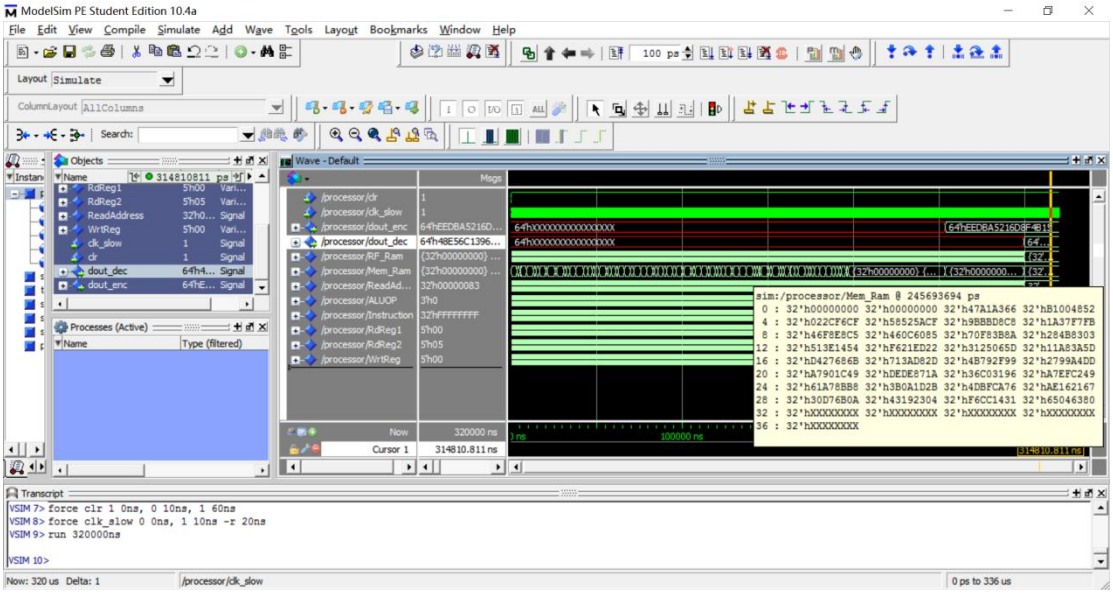
The Decoder successfully identifies this instruction as an I-type SHL, and outputs the Op code signal as LEFT SHIFT and will be passed into ALU unit.

Checkpoint: ALU Unit



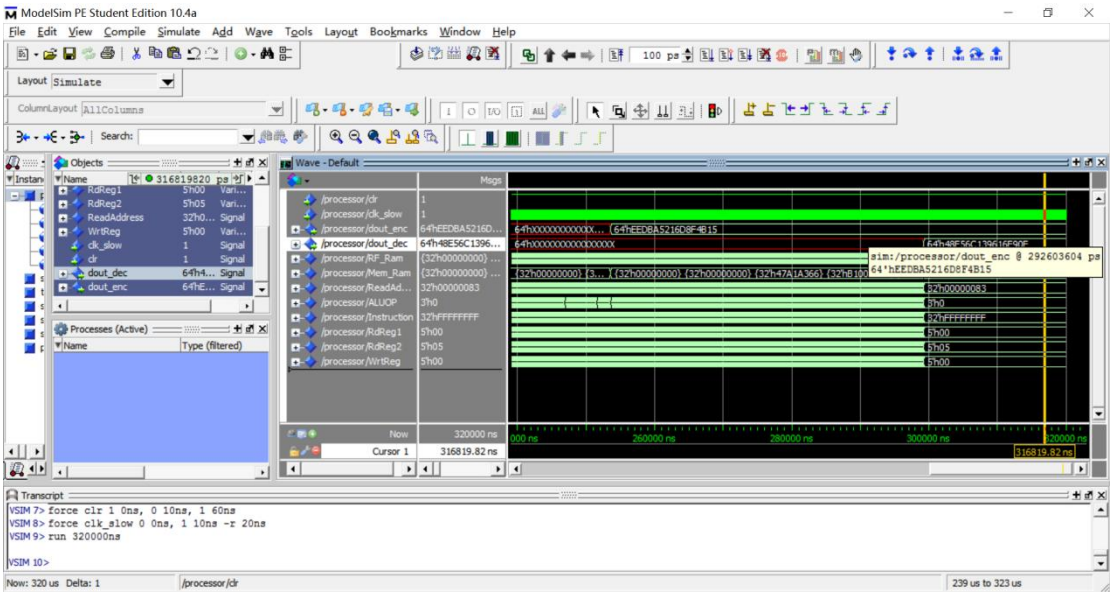
The ALU reads one input from register R3 with value 0xB7E15163 and shifts with immediate value 0x3, then stores the result into register R13.

- **Processor Design**



Data Mem After Key Expansion

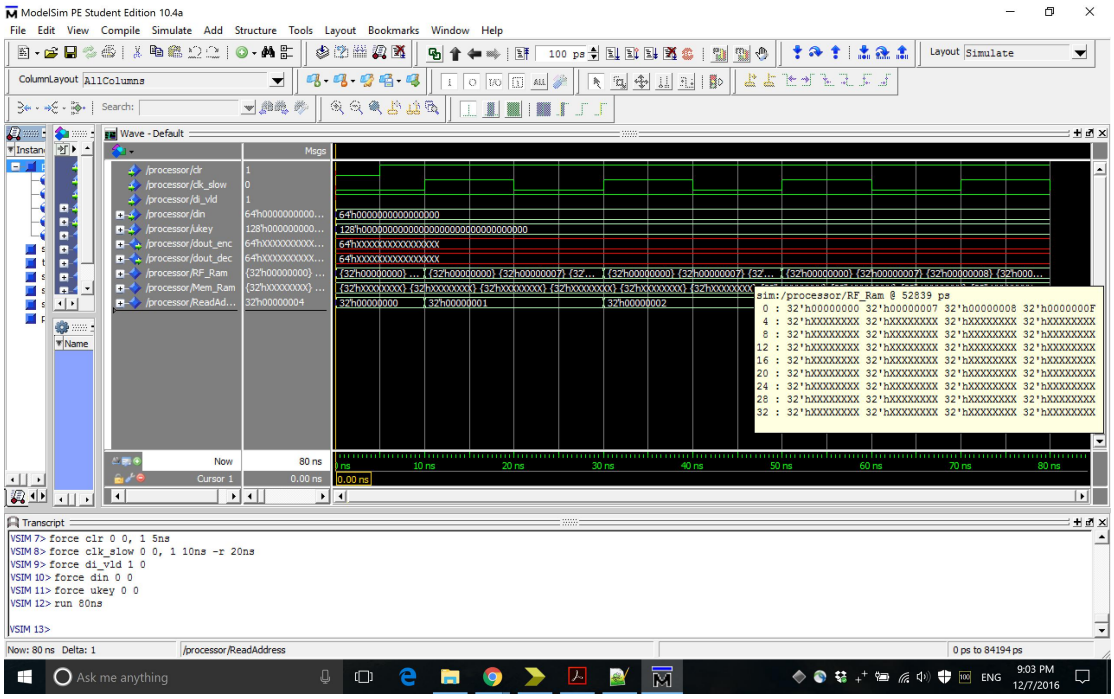
After 78 rounds of key expansion, the data memory holds the whole skey from address 0x6 to 0x31. The demo ukey here is 128'b0. The skey is exactly the same as the skey generated by RC5 key expansion tested in Lab7.



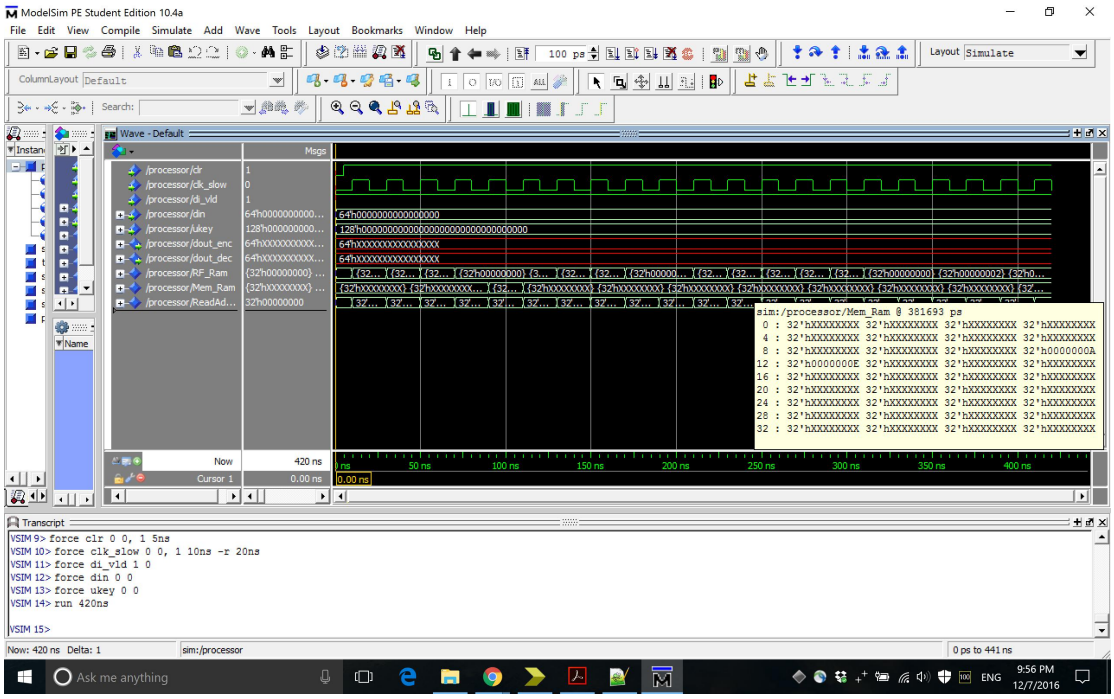
ENC/DEC

After the key expansion, both ENC and DEC output have been generated. The results comply with the result of Lab 7.

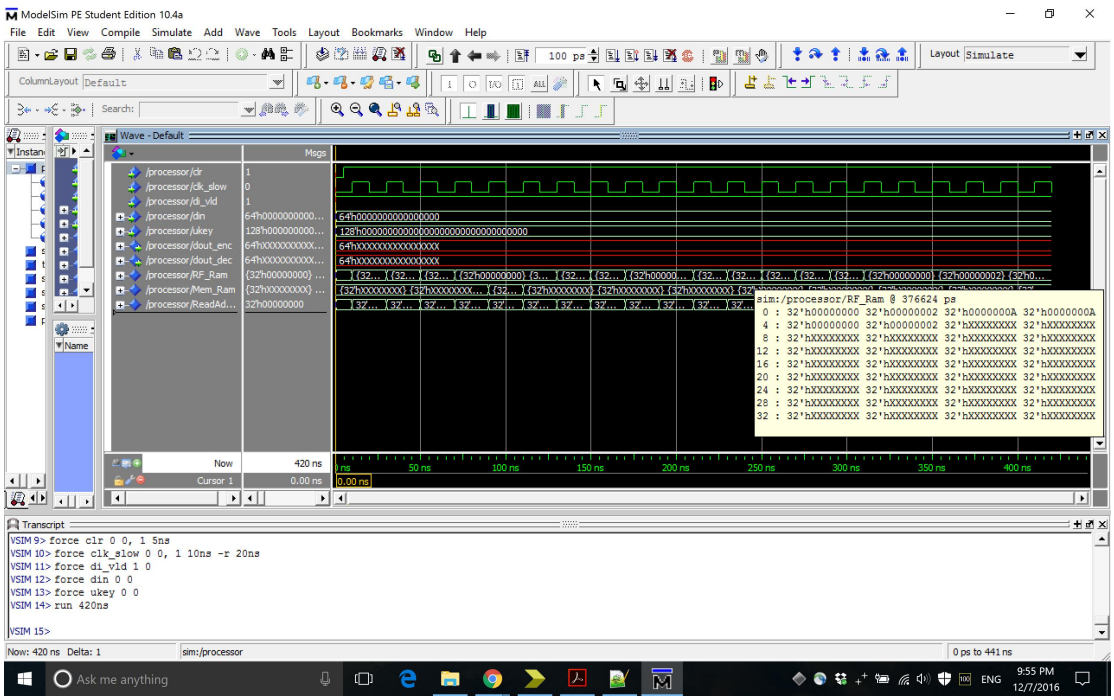
- **Sample Program Tests**



RF of Sample Program 1



Data Memory of Sample Program 2



RF of Sample Program 2

- **Assembly Code**

See Appendix for assembly code of key expansion, encryption and decryption.

- **High Level Description**

The entire RC5 process is implemented within the Processor component of our project.

Key Expansion:

Before the key expansion starts, the Instruction Memory and Data Memory have been initialized by final_inout components. The Ins Mem is filled with the machine code of translated assembly program to implement key expansion algorithm. The ukey is fed through top-level port map, and the first 4 addresses in the Data Mem are used to store the 128 bit ukey.

Encryption:

After the key expansion process, the Data Mem has been loaded with 26 keys starting from address 6 to 31. The encryption process will be proceeded. Similarly, the machine code of translated assembly program to implement RC5 encryption algorithm has been loaded into Ins Mem at the very beginning. Then encrypted text will be stored in the address 32 and 33 of the Data Mem.

Decryption:

After the encryption process, the decryption will be proceeded. The instructions have been loaded at the very beginning and the deciphered text will be stored in the address 34 and 35 of the Data Mem.

- **Processor Interfaces**

In our project, all 16 switches are used as input. Push-button <P17> is used to feed the input to ukey. 4 LEDs <V15>, <V14>, <V12>, <V11> are used to indicate first, second, third and the last 32-bit value of ukey. Similarly, push-button <N17> is used to feed the input to plain text or ciphered text. 2 LEDs are used to indicate 2 32-bit value. Push-button <C12> is defined as CLR signal. Push-button <P18> is used to select encryption or decryption output. Push-button <M17> is used to generate output signal on 7-segment LEDs.

After generating the output, the switch <V10> can be turned on to show single stepping output of each instruction. By pushing push-button<M18>, the 7-segment LED is going to show every round of output of each instruction. If current instruction is an ALU instruction, then the ALU result is going to be displayed on the 7-segment LED; If current instruction is an LOAD/STORE instruction, the content of the LOAD/STORE value is going to be displayed.

- **Performance and area analysis**

Synthesis report:

Totally 5162 out of 126800 (4%) Slice Registers and totally 5935 out of 63400 (10%) Slice LUTs are utilized and all of them are used as logic units. 9913 LUT Flip Flop pairs are designed. Among them 4751 (47%) are unused Flip Flop, 3978 (40%) are unused LUTs and 1184 (11%) are fully used LUT-FF pairs. The number of unique control sets is 723. Totally 44 out of 210 (15%) bonded IOBs are used.

Place and Route Report:

Totally 5162 out of 126800 (3%) Slice Registers are utilized, among which 4970 are used as Flip Flop and the rest 192 are latches. The number of O6 output only is 4859. The number of O5 output only is 39. The number of O5 and O6 output is 496, and 83 more as route-thrus, so totally making 5477 out of 63400 slice LUTs.

Totally 2894 out of 15850 (18%) occupied Slices and 8249 LUT Flip Flop pairs are used. Among them 3207 (38%) are unused Flip Flop, 2772 (33%) are unused LUTs and 2270 (27%) are LUT-FF pairs. Totally 44 out of 210 (15 %) bonded IOBs are used.

Minimum period is 7.279ns, providing maximum frequency of 137.373Mhz.

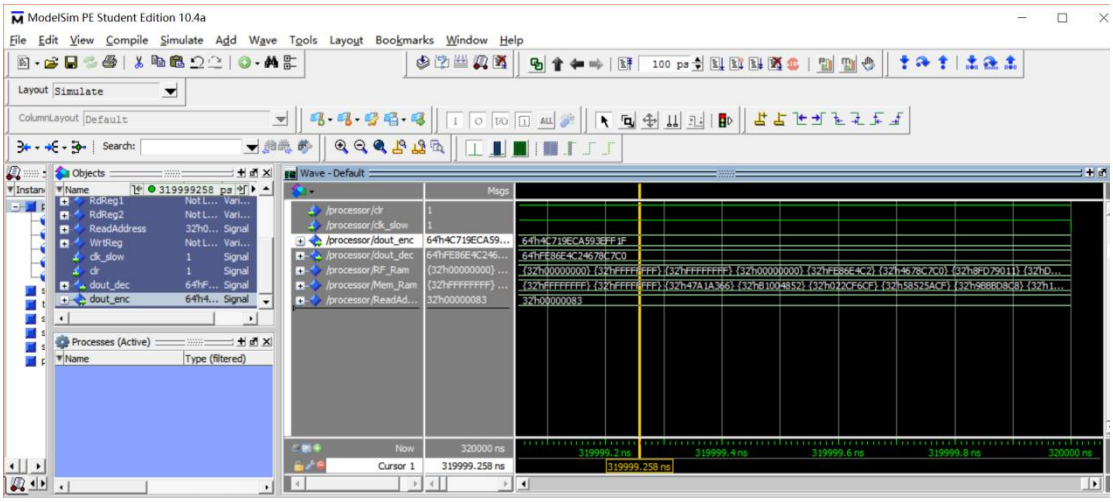
- **Overall Design Verification**

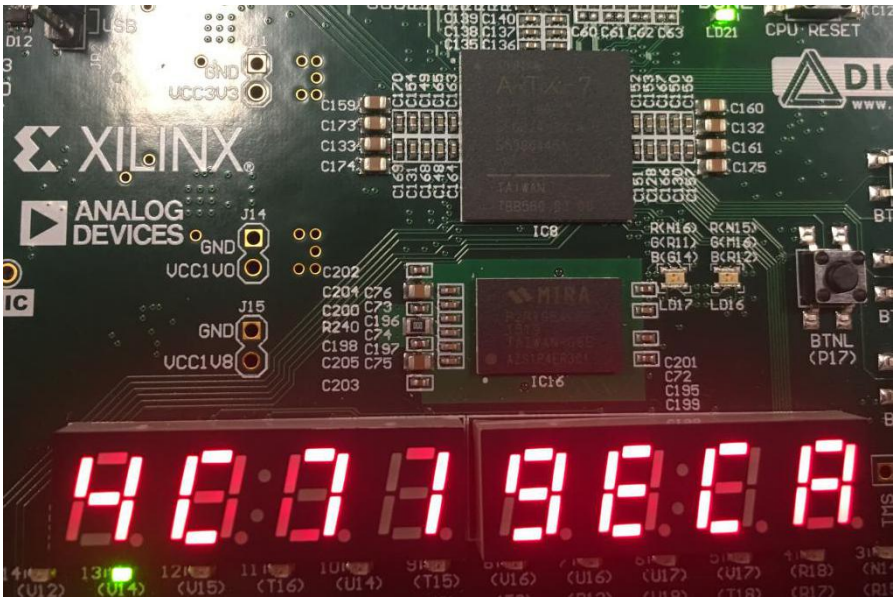
Several test cases are used to verify the accuracy of our project. Multiple combinations of values of key and the plain text/ciphered text and used to compare with the result generated from RC5-dedicated program.

Case 1: ukey = 128'b0, din = 64b'1

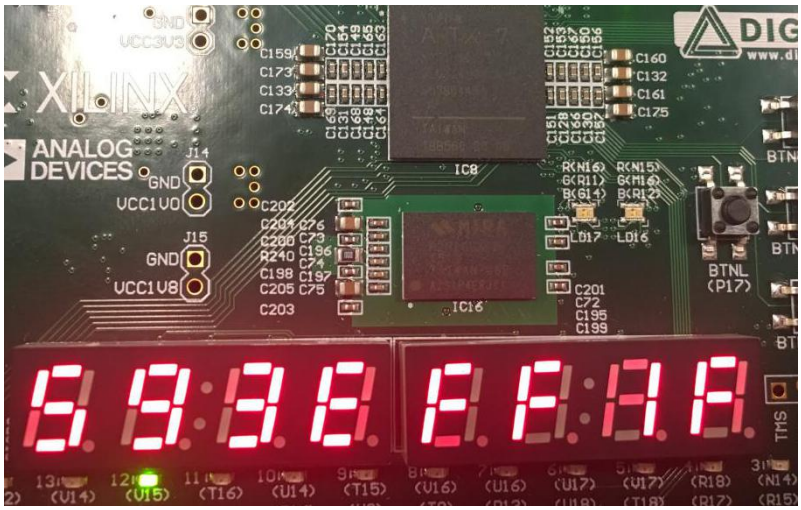
The encryption result is 0x4C719ECA593EFF1F;

The decryption result is 0xFE86E4C24678C7C0

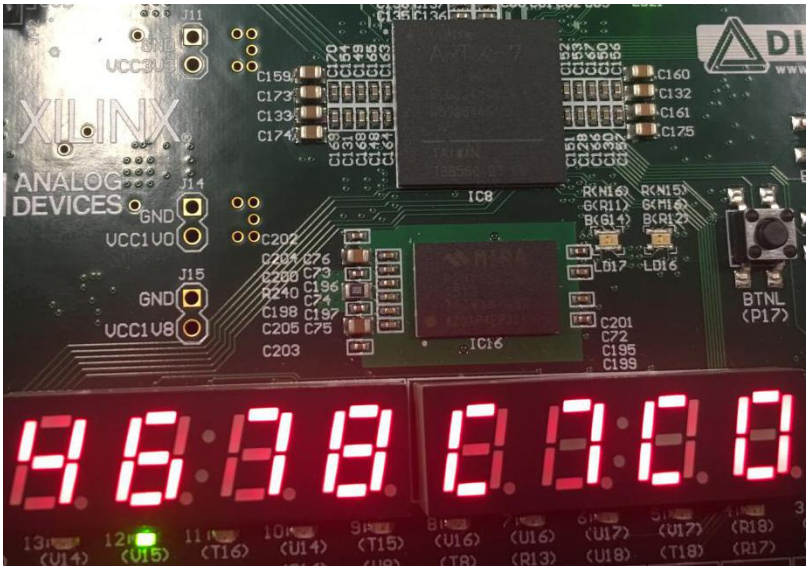




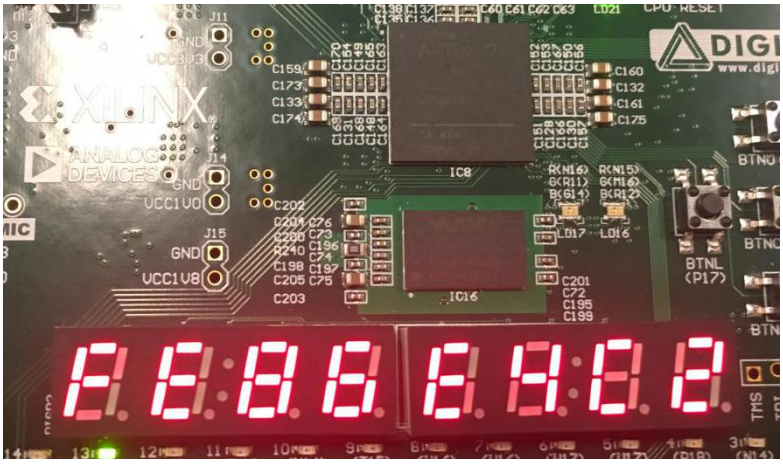
Higher 32-bit of enc_dout



Lower 32-bit of enc_dout



Higher 32-bit of dec_dout

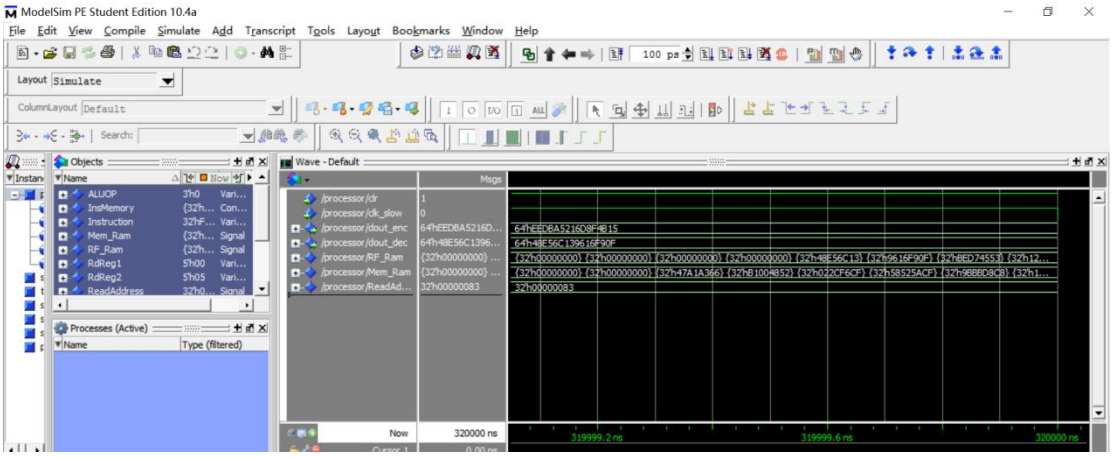


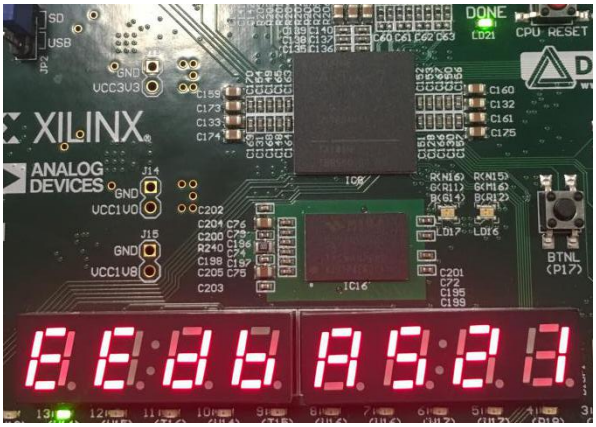
Lower 32-bit of dec_dout

Case 2: ukey = 128'b0, din = 64b'0

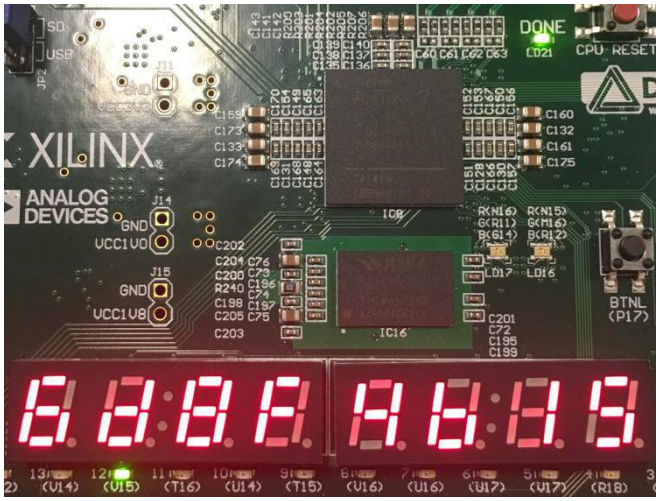
The encryption result is 0xEEDBA5216D8F4B15;

The decryption result is 0x48E56C139616F90F;

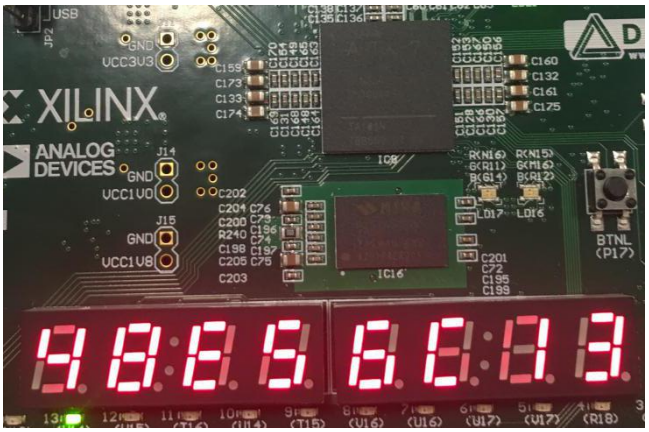




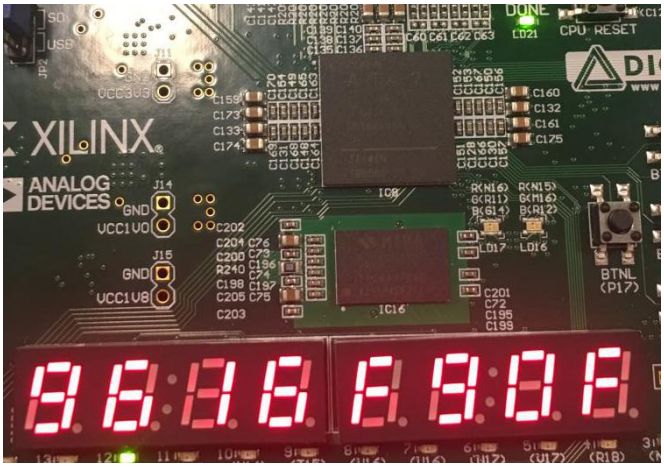
Higher 32-bit of enc_dout



Lower 32-bit of enc_dout



Higher 32-bit of dec_dout

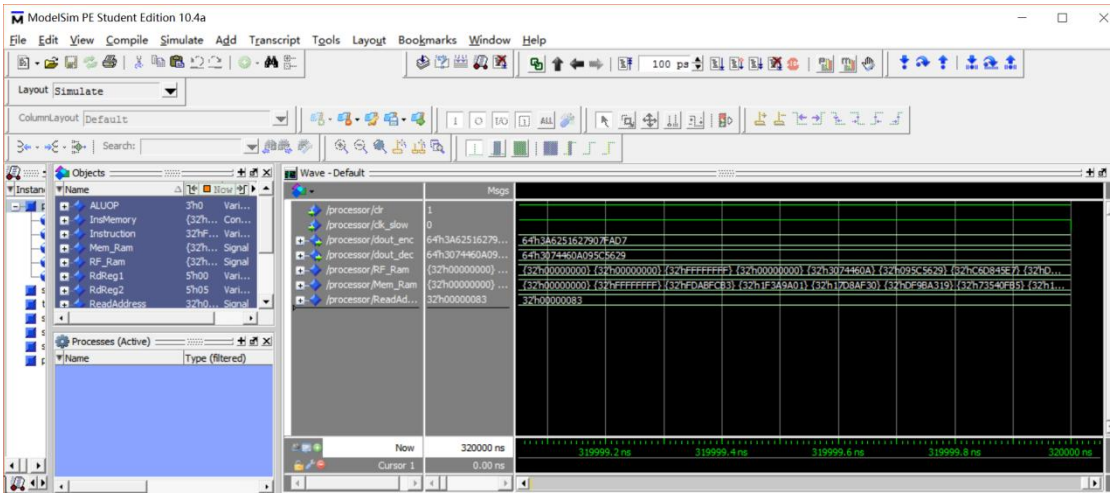


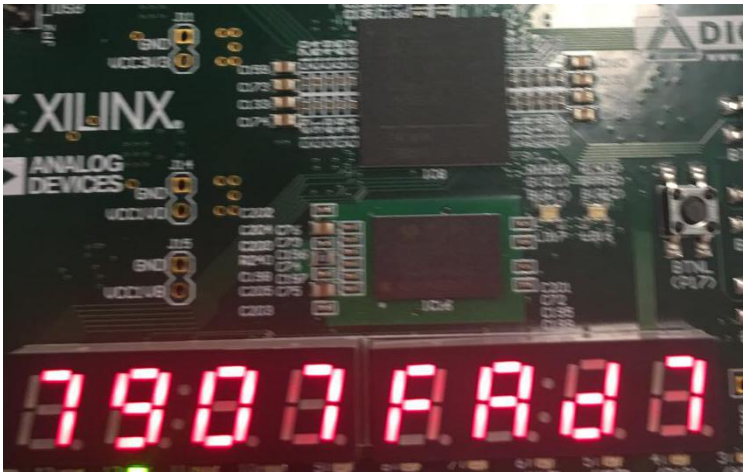
Lower 32-bit of dec_dout

Case 3: ukey = 0x000000000000000000000000FFFFFFFF, din = 0x00000000FFFFFFFF

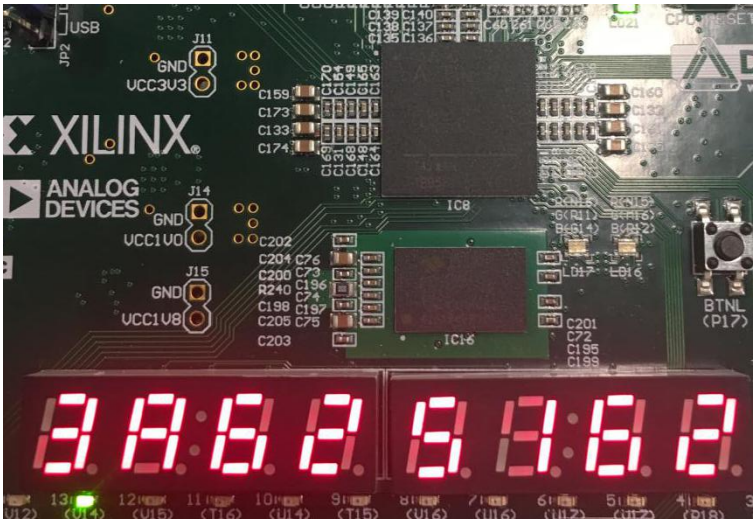
The encryption result is 0x3A6251627907FAD7;

The decryption result is 0x3074460A095C5629;

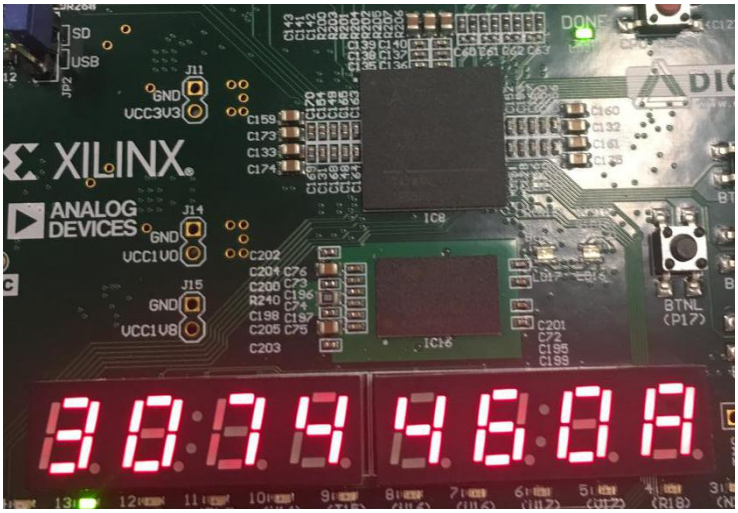




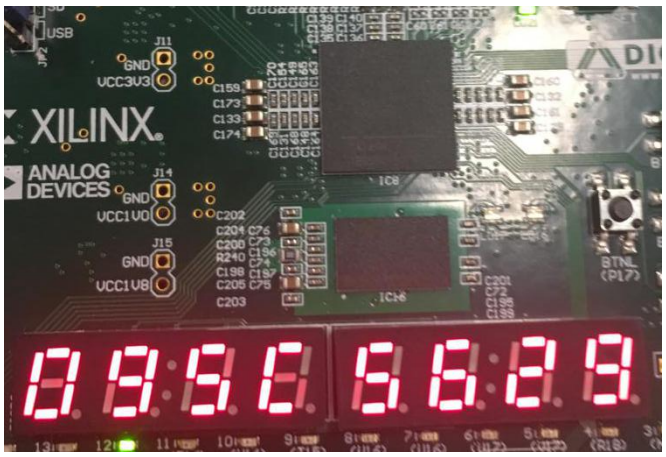
Higher 32-bit of enc_dout



Lower 32-bit of enc_dout



Higher 32-bit of dec_dout



Lower 32-bit of dec_dout

- **Appendix**

Key Expansion:

```
ADD R0 R0 R10
ADD R0 R0 R8
ADD R0 R0 R9
ADD R0 R0 R1
ADD R0 R0 R2
ADDI R0 R17 78
ADDI R0 R18 26
ADDI R0 R19 4
LW R8 R11 S_ADD--------RETURN
ADD R11 R1 R11
ADD R11 R2 R3
SHL R3 R13 3
SHR R3 R14 29
ADD R13 R14 R4
ADD R4 R2 R5
LW R9 R12 L_ADD
ADD R12 R5 R6
// R13 COUNTER R14 INDICATE SHIFT
ANDI R6 R14 000…11111
ADD R0 R0 R13
ADD R0 R6 R7
BEQ R13 R14 IMM----------IMM=SW R8 R4 S_ADD
SHL R7 R15 1
SHR R7 R16 31
ADD R15 R16 R7
ADDI R13 R13 1
JUM IMM----------IMM=BEQ R13 R14 IMM
SW R8 R4 S_ADD
SW R9 R7 L_ADD
ADDI R8 R8 1
ADDI R9 R9 1
ADDI R10 R10 1
BNE R8 R18 IMM----------IMM=BNE R9 R19 IMM
ADD R0 R0 R8
BNE R9 R19 IMM----------IMM=BNE R10 R17 IMM
ADD R0 R0 R9
BNE R10 R17 IMM----------IMM=LW R8 R11 S_ADD
OUTPUT
```

RC5 Encryption:

```
//R1 adin R2 bdin
LW R0 R1 adin_offset
LW R0 R2 bdin_offset
//R3 COUNTER
ADD R0 R0 R3
ADDI R0 R30 26
// R4= A_REG       R5=B_REG
ADD R0 R0 R4
ADD R0 R0 R5
LW R3 R31 OFFSET----------OFFSET=round key address in data memory
ADD R4 R31 R4
LW R3 R31 OFFSET+1
ADD R5 R31 R5
ADDI R3 R3 2

NOR R4 R0 R6                    //RETURN
NOR R5 R0 R7
AND R4 R7 R7
AND R5 R6 R6
//R8=AB_XOR
ADD R6 R7 R8

//R6 COUNT    R7 INDICATE SHIFT NUM
ANDI R5 R7 000…11111
ADD R0 R0 R6
ADD R0 R8 R9

BEQ R6 R7 IMM----------IMM=LW R3 R31 OFFSET
SHL R9 R10 1
SHR R9 R11 31
ADD R10 R11 R9
ADDI R6 R6 1
JMP IMM----------IMM=BEQ R6 R7 IMM

LW R3 R31 OFFSET
ADD R4 R31 R4

NOR R4 R0 R6
NOR R5 R0 R7
AND R4 R7 R7
AND R5 R6 R6
//R12=BA_XOR
```

```
ADD R6 R7 R12

//R6 COUNT    R7 INDICATE SHIFT NUM
ANDI R4 R7 000…11111
ADD R0 R0 R6
ADD R0 R12 R13

BEQ R6 R7 IMM----------IMM=LW R3 R31 OFFSET+1
SHL R13 R10 1
SHR R13 R11 31
ADD R10 R11 R13
ADDI R6 R6 1
JMP IMM----------IMM=BEQ R6 R7 IMM

LW R3 R31 OFFSET+1
ADD R5 R31 R5
ADDI R3 R3 2

BLT R3 R30 IMM----------IMM=NOR R4 R0 R6
```

RC5 decryption:

```
//LW ADIN BDIN
LW R0 R1 adin_offset
LW R0 R2 bdin_offset
ADDI R0 R3 24
LW R3 R8 OFFSET+1      ------RETURN
SUB R5 R8 R7
//R8 COUNTER R9 INDICATE SHIFT NUM
ANDI R4 R9 000….11111
ADD R0 R0 R8
ADD R0 R7 R11

BEQ R8 R9 IMM----------IMM=NOR R4 R0 R8
SHR R11 R12 1
SHL R11 R13 31
ADD R12 R13 R11
ADDI R8 R8 1
JUM IMM----------IMM=BEQ R8 R9 IMM

NOR R4 R0 R8
NOR R11 R0 R9
AND R4 R9 R9
AND R8 R11 R8
ADD R8 R9 R5

LW R3 R8 OFFSET
SUB R4 R8 R6

//R8 COUNTER R9 INDICATE SHIFT NUM
ANDI R5 R9 000….11111
ADD R0 R0 R8
ADD R0 R6 R10

BEQ R8 R9 IMM----------IMM=NOR R10 R0 R8
SHR R10 R12 1
SHL R10 R13 31
ADD R12 R13 R10
ADDI R8 R8 1
JUM IMM----------IMM=BEQ R8 R9 IMM

NOR R10 R0 R8
NOR R5 R0 R9
AND R10 R9 R9
```

```
AND R5 R8 R8
ADD R8 R9 R4
SUBI R3 R3 2

BNE R3 R0 IMM----------IMM=LW R3 R8 OFFSET+1
LW R3 R8 OFFSET+1
SUB R5 R8 R5
LW R3 R8 OFFSET
SUB R4 R8 R4
OUTPUT
```