CAIRO UNIVERSITY
FACULTY OF ENGINEERING
ELECTRONICS & ELECTRICAL
COMMUNICATIONS DEP.

FFE EQUALIZER

Digital Design (ELC4007)

Names	ID
ابانوب عطية سعيد	9202026
أحمد ابراهيم محمد عبد المعطى	9202045
محمد محسن كمال موسي	9203326
محمود شريف محمود حسن	9203409
فاطمة عادل شعبان	9203047
بسمه وليد حسنى ابوهاشم عشرى	9202383

Submitted to: Dr. Karem Osama, Eng. Mahmoud Yehia.

Introduction

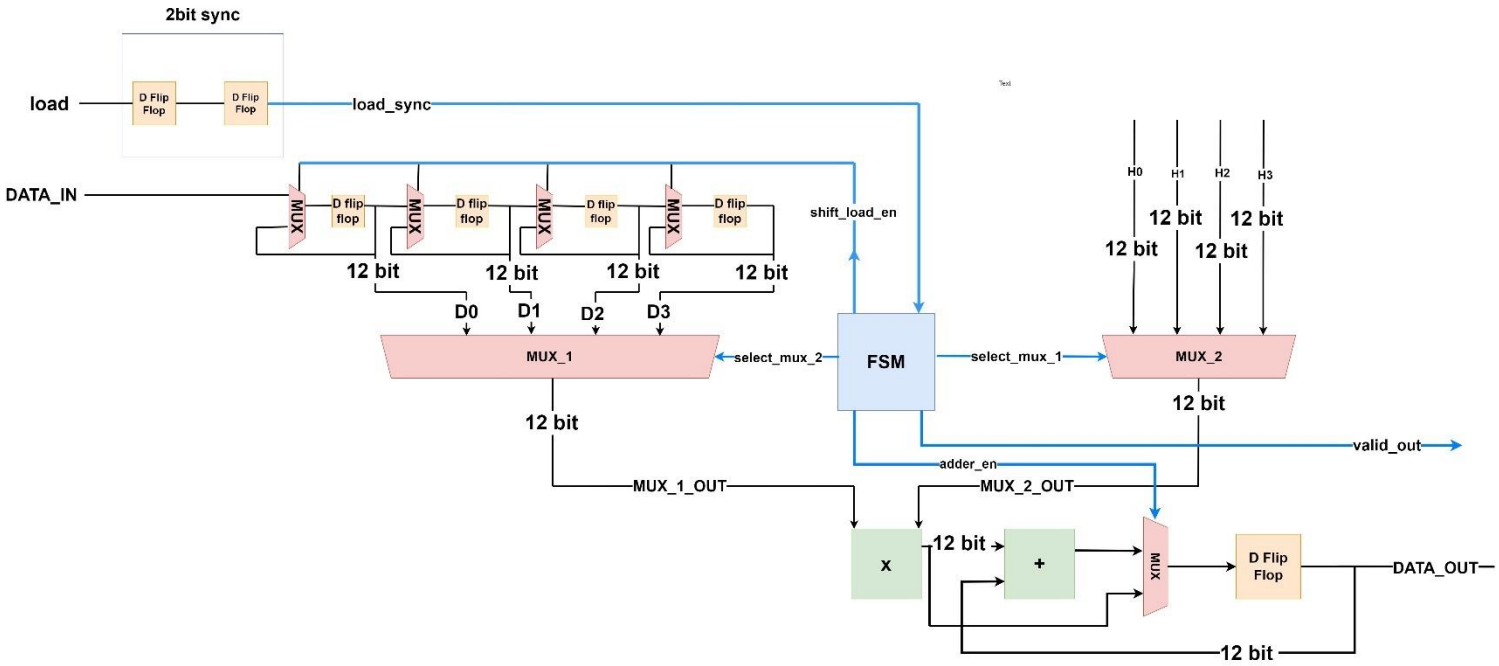
A Feed-Forward Equalizer (FFE) is essential in high-speed digital communication for combating inter-symbol interference (ISI), which distorts signals during transmission. By applying adjustable coefficients to the incoming signal, an FFE reshapes it to counteract distortions, improving signal integrity and enabling higher data rates.

Key aspects of FFE include:

- **Taps and Coefficients**: Adjustable weights applied to signal samples to minimize ISI.
- **Adaptive Algorithms**: Real-time adjustment of coefficients using algorithms like LMS and RLS.
- **Performance Metrics**: Measured by improvements in bit error rate (BER) and signal-to-noise ratio (SNR).

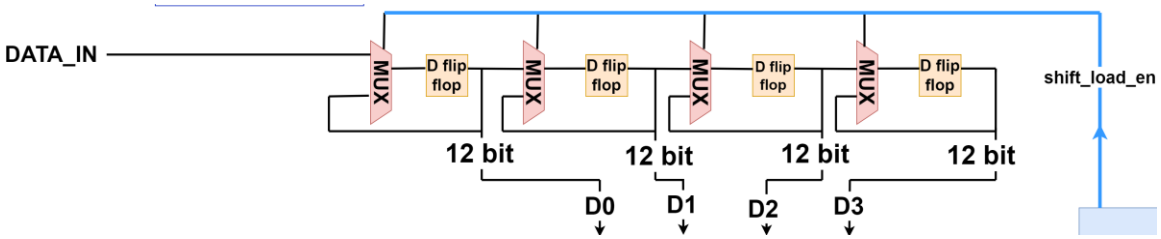
FFEs are crucial in standards like Ethernet, USB, PCIe, and optical fiber communications, ensuring reliable and efficient data transmission over challenging channels.

Hardware Design of FEE



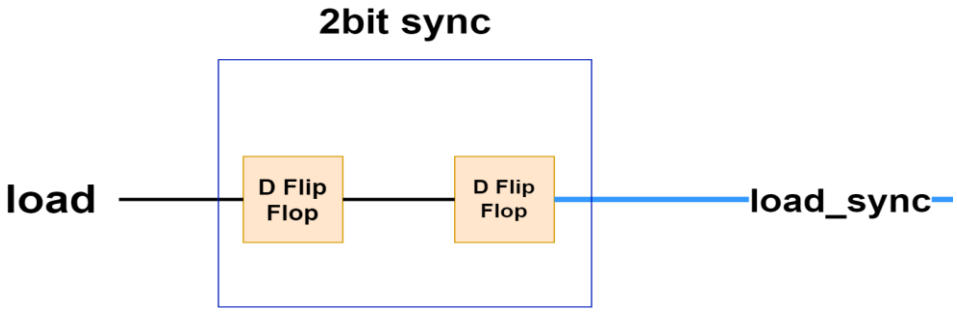
Sub-Blocks Description

- **Shift Register:**



Port	Direction	Width	Description	Connected to
DATA_IN	IN	12	Input data comes from TX	Top input port
Shift_load_en	IN	1	Control signal to enable shift operation	FSM
D0	OUT	12	Registered Values	MUX_1
D1	OUT	12	Registered Values	MUX_1
D2	OUT	12	Registered Values	MUX_1
D3	OUT	12	Registered Values	MUX_1

- **Two-bit synchronizer**

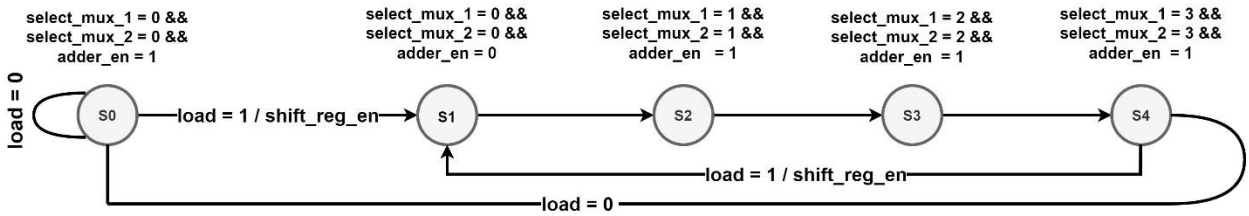


Port	Direction	Width	Description	Connected to
Load	IN	1	Input signal comes from TX	Top input port
Load_sync	OUT	1	Synchronize version of Load signal	FSM

- **Multiplier**: We use a 12-bit multiplier that will multiply at each clock cycle of the FFE one of the data words by one of the FFE tabs. After doing Fixed Point Analysis, we chose to divide those 12-bit operands to 6-bits for the integer part and 6-bits for fractions. The output would be 24-bits, and we will take only 12-bits from bit (6) to bit (17) to represent the output, 6-bits of them from bit(6) to bit(11) represent the fractional part, and the rest from bit(12) to bit(17) represent the integer part of the output.

- **Adder**: We have a 12-bit adder, the output each time would be 13-bits, but we truncate the Least Significant Bit. After receiving the load signal, the adder starts by adding the multiplier output to 0 in the first clock cycle and in the next cycles it begins accumulating the multiplier outputs to each other.

- **FSM**:



- **Note:**
  - For each Flip Flop there is port for **clock** and port for **Reset**
  - The **clock** of all Flip Flop is the FFE clock frequency (4 MHz)
  - All internal signals are signed signal with 12 bit width
  - **adder_en** signal decided if Register on output port will store output of Adder or output of Multiplier

Pseudo Code

ALGORITHM 1 : ALGORITHM TO PERFORM FFE EQUATION $y_i = \sum_{j=0}^{j=3} h_j D_{i-j}$

```
1   INPUT  DATA_IN : Data come from TX
2   INPUT load : Signal come from TX to ensure the validity of the data
3   INPUT clk : clock signal
4   INPUT rst : reset signal
5   OUTPUT  DATA_OUT : Result from FIR
6   INITIALIZE : D0 ,D1 ,D2 ,D3 , valid_out and DATA_OUT to zero

7   AT every positive edge of clk
8       IF rst is active low  THEN
9           SET current_state = S0
10      ELSE
11          SET next_state to current_state
12      ENDIF
13  ENDAT

14  SWITCH   current_state
15      CASE S0 :
16          IF load is low THEN
17              SET next_state = S0
18          ELSE
19              SET next_state = S1
20              SET shift_load_en = 1
21          ENDIF
22      CASE S1 :
23          CALCULATE  DATA_OUT = D0 * H0 + 12'b0
24          SET next_state = S2
25      CASE S2 :
26          CALCULATE  DATA_OUT = D1*H1 + D0 * H0
27          SET next_state = S3
28      CASE S3 :
29          CALCULATE  DATA_OUT = D2*H2 + D1*H1 + D0 * H0
30          SET next_state = S4
31      CASE S4 :
32          CALCULATE  DATA_OUT = D3*H3 + D2*H2 + D1*H1 + D0 * H0
33          IF load is low THEN
34              SET next_state = S0
35          ELSE
36              SET next_state = S1
37              SET shift_load_en = 1
34  ENDSWITCH
```

Pseudo Code (More Details)

```
//_____________Top System _____________:
inputs: load,Data_in[12],CLK,RST

outputs:Data_out[12] ,valid_output


// values for filter taps
h0=12'b 000000100000
h1=12'b 111111110000
h2=12'b 000000001010
h3=12'b 111111111100




//******** BIT_SYNCHONIZER************

    input    CLK
    input    RST
    input    load
    output   load_sync

//LOGIC design:
inernal flipflop reg_sync[2]

every +ve CLK or -ve RST

  if(!RST)
    {
      reg_sync=0
    }


  else
   {
    reg_sync[0]=reg_sync[1]
    reg_sync[1] = load
   }
```

```
we make this connectection :
load_sync =  reg_sync[0]




//***************shift_reg *********

    input             CLK
    input             RST
    input             shift_load_en
    input              Data_in
    output             shift_reg_out[48]


//LOGIC design:
every +ve CLK or -ve RST {
if (!RST)
{
    shift_reg_out = 0
}

else if (shift_load_en){
    //SHIFT RIGHT & CAPTURE Inputs
    shift_reg_out[11:0]  = shift_reg_out[23:12]
    shift_reg_out[23:12] = shift_reg_out[24:35]
    shift_reg_out[24:35] = shift_reg_out[47:36]
    shift_reg_out[47:36] = Data_in

}
else
    // retain old value
    shift_reg_out =shift_reg_out

}


//***************MUX1*************:
inputs: h0,h1,h2,h3,select_mux_1

output:h_i[12]
```

```
//LOGIC design:
case(select_mux_2)
0: h_i= h0
1: h_i= h1
2: h_i= h2
3: h_i= h3


//***************MUX2*************:
inputs: shift_reg_out[11:0],shift_reg_out[23:12],
shift_reg_out[24:35],shift_reg_out[47:36],select_mux_2


output: D_i[12]

//LOGIC design:
case(select_mux_2)
0: D_i= shift_reg_out[11:0]
1: D_i= shift_reg_out[23:12]
2: D_i= shift_reg_out[24:35]
3: D_i= shift_reg_out[47:36]



//******************* multilpier block *************
    input    D_i,h_i
    output  mul_out[24]


    //LOGIC design:
    while(1){
        mul_out = D_i*h_i
    }



//************ adder_block************

        input    CLK
        input   RST
        input   adder_en
        input    mul_out[11:0],Data_out_ff   //  we  trim  MSB  from
```

```
multilpier output
         output   Data_out


//LOGIC design:
every +ve CLK edge or -ve RST edge {
     if(!RST)
          Data_out=0
     else
     {
          if(adder_en)
          {
               Data_out = mul_out + [11:0],Data_out_ff
          }
          else
          {
               Data_out=mul_out
          }


     }

}



//**************Finite State Machine************************//
     input    CLK
     input   RST
     input   load_sync
     output  adder_en,valid_output
     output   select_mux_1
     output  select_mux_2
     output  shift_load_en



//we  start  at  initial  state  s0  (current_state  =  s0)  until
Load_Sig is asserted

//every clock edge we make current_state = next_state

// also we go to s0 on a -ve edge reset
```

```
case(current_state)
s0:
// ctrl signals :
adder_en          = 0
select_mux_1      = 0
select_mux_2      = 0
valid_output = 0

if(load_sync == 1) {
next_state        = s1
shift_load_en     = 1

}
else{
next_state        = s0
shift_load_en     = 0
}



s1:
// ctrl signals :
select_mux_1      = 0
select_mux_2      = 0
shift_load_en     = 0
valid_output = 0


next_state        = s2

s2:
// ctrl signals:

select_mux_1      = 1
select_mux_2      = 1
shift_load_en     = 0
valid_output = 0

next_state        = s3

s3:
```

```
// ctrl signals:
select_mux_1     = 2
select_mux_2     = 2
shift_load_en    = 0
valid_output = 0


next_state       = s4

s4:
// ctrl signals :
select_mux_1     = 3
select_mux_2     = 3
valid_output    = 1

if(load == 1){
    next_state = s1
    shift_load_en = 1
} else {
    Next_state = s0
    Shift_load_en = 0
}
```

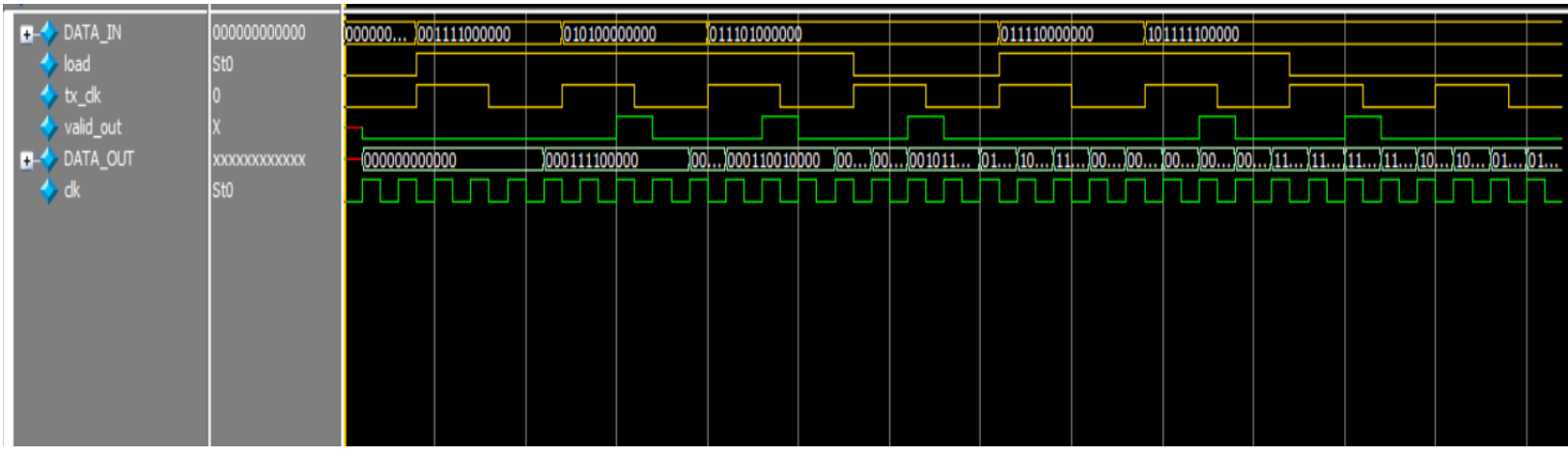
Verilog Simulation

```
# input data : 15
# output is 7.500000
# result should be 7.500000
#
# ----------------------------------------
#
# input data : 20
# output is 6.250000
# result should be 6.250000
#
# ----------------------------------------
#
# input data : 29
# output is 11.843750
# result should be 11.843750
#
# ----------------------------------------
#
# input data : 30
# output is 9.937500
# result should be 9.937500
#
# ----------------------------------------
#
# input data is -16.500000
# output is -12.468750
# result should be -12.468750
```

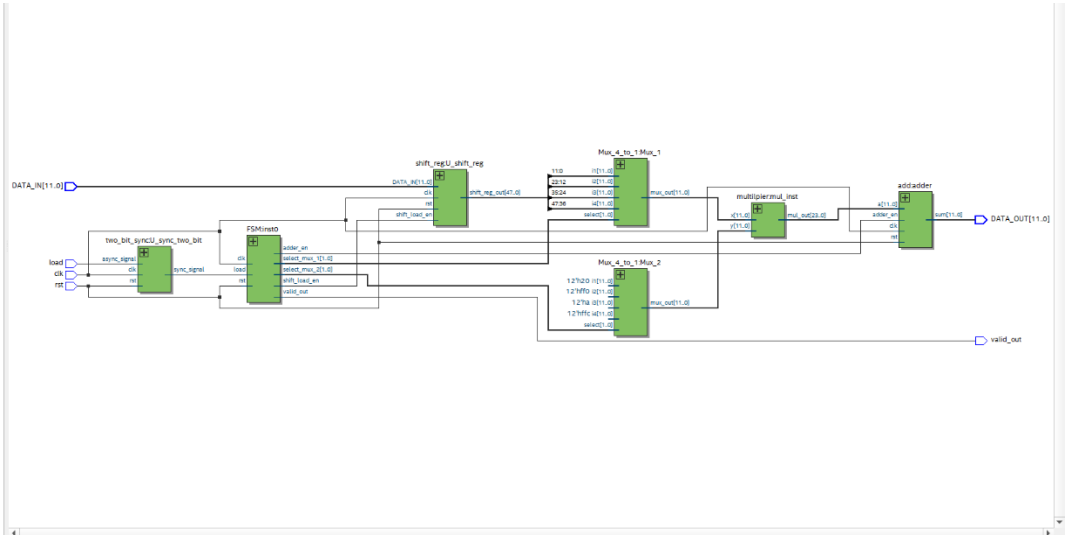
- $y_i = -12.468750$

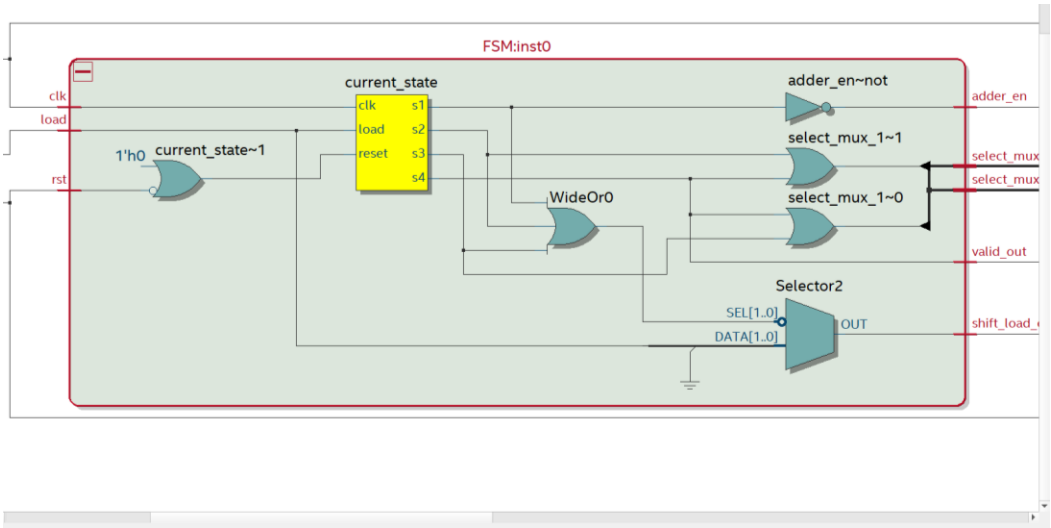
by apply equation $y_i = \sum_{j=0}^{j=3} h_j D_{i-j}$:

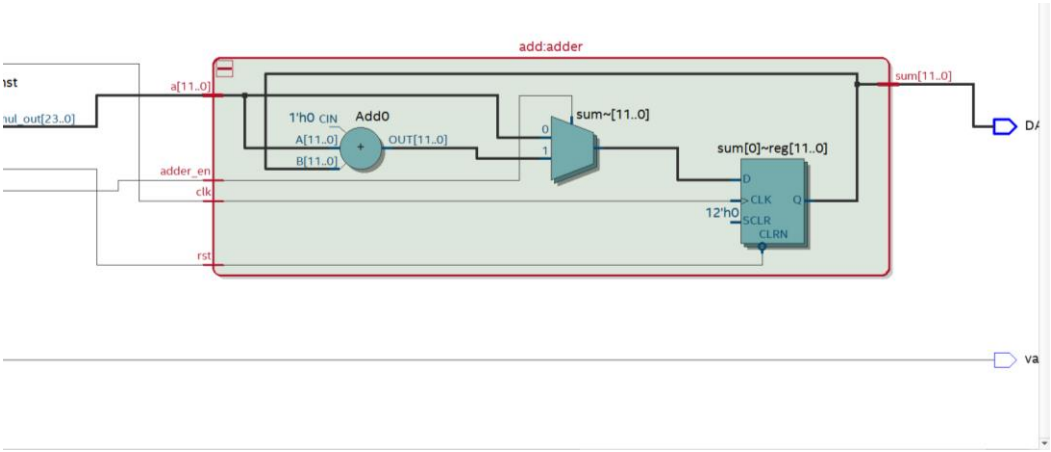
- **For first input (15):**
  - $y_i = 15 * h0 = 15 * 0.5 = \ 7.5$
- **For second input (20) :**
  - $y_i = 20 * h0 + 15 * h1$
  - $y_i = 20 * 0.5 - 15 * 0.25 = 6.25$
- **For third input (29) :**
  - $y_i = 29 * h0 + 20 * h1 + 15 * h2$
  - $y_i = 29 * 0.5 - 20 * 0.25 + 15 * 0.15625$
  - $y_i = 11.84375$
- **For fourth input (30) :**
  - $y_i = \ 30 * h0 + 29 * h1 + 20 * h2 + 15 * h3$
  - $y_i = 30 * 0.5 - 29 * 0{,}25 + 20 * 0.15625 - 15 * 0.0625$
  - $y_i = 9.9375$
- **For fifth input (-16) :**
  - $y_i = \ -16 * h0 + 30 * h1 + 29 * h2 + 20 * h3$
  - $y_i \ = -16 * 0.5 - 30 * 0{,}25 + 29 * 0.15625 - 20 * 0.0625$



Synthesis:


DATA_IN[11..0]
shift_reg:U_shift_reg
Mux_4_to_1:Mux_1
multiplier:mul_inst
add:adder
DATA_OUT[11..0]
two_bit_syncU_sync_two_bit
FSM:inst0
load
clk
rst
Mux_4_to_1:Mux_2
valid_out


FSM:inst0
clk
load
rst
current_state
clk
load
reset
s1
s2
s3
s4
1'h0
current_state~1
adder_en~not
adder_en
select_mux_1~1
select_mux
select_mux_1~0
WideOr0
valid_out
Selector2
SEL[1..0]
DATA[1..0]
OUT
shift_load_


add:adder
a[11..0]
adder_en
clk
rst
1'h0
CIN
Add0
A[11..0]
B[11..0]
OUT[11..0]
sum~[11..0]
sum[0]~reg[11..0]
D
CLK
Q
12'h0
SCLR
CLRN
sum[11..0]