```vhdl
library ieee;
	use ieee.std_logic_1164.all;
	use ieee.numeric_std.all;
	use ieee.std_logic_unsigned.all;

--Declaring inputs and outputs
	entity AES_Algorithm is
		Port(
		--input signals
		--user defined key
		KEY_IN : in  STD_LOGIC_VECTOR (127 downto 0);
		--plaintext that is input
		D_IN : in  STD_LOGIC_VECTOR (127 downto 0);
		--reset and clock pulses used to control itterations
		RESET : in  STD_LOGIC;
		CLOCK : in  STD_LOGIC;
		LOAD : in  STD_LOGIC;
		--cypher text that is output after completion
		D_OUT : out STD_LOGIC_VECTOR(127 downto 0);
		--output signal to signifie completion of encryption
		COMPLETE : out STD_LOGIC;
		GOOD : out  STD_LOGIC);
	end AES_Algorithm;


	architecture Behavioral of AES_Algorithm is
		--signals and types that are used in the key generation module
		signal NewWord : std_logic_vector(31 downto 0);
		signal RoutineWord  : std_logic_vector(31 downto 0);
		signal SubRoutineWord : std_logic_vector(31 downto 0);
```

```
			--Word array to hold values of the user defined input key to generate new key for each round
			type WordArray is array(0 to 3) of std_logic_vector(31 downto 0);
			signal Word32 : WordArray;
			--Round conctant array to store the round constant values
			type RoundConstantArray is array(0 to 11) of std_logic_vector(7 downto 0);
			--The round constant values are derived from the galois field transformation and are predefined
			constant RoundConstantV : RoundConstantArray := (x"01",x"02",x"04",x"08",x"10",x"20",x"40",x"80",x"1b",x"36",x"aa",x"bb");
			--Signals that are used to interconnect the modules
			signal NewKey : std_logic_vector(127 downto 0);
			signal ADDKey : std_logic_vector(127 downto 0);
			signal BYTESUB : std_logic_vector(127 downto 0);
			signal RowShift : std_logic_vector(127 downto 0);
			signal NewRound : std_logic_vector(127 downto 0);
			signal MixedColumnOut : std_logic_vector(127 downto 0);
			type stateRowT is array (0 to 3) of std_logic_vector(7 downto 0);
   --Declaring signals and the type that are used in the row shifting module.
   type Matrix_array is array (15 downto 0) OF std_logic_vector(7 downto 0);
   signal Input_Matrix, Shifted_Matrix : Matrix_array;
			--Round counter signal for state machine
			signal RoundNum : std_Logic_vector(5 downto 0);
			--	signal key : std_logic_vector (127 downto 0) := x"2b7e151628aed2a6abf7158809cf4f3c";
			-- signal plain : std_logic_vector (127 downto 0) := x"3243f6a8885a308d313198a2e0370734";


	begin


--The Key creator mdule creates a new cipher key that stems from the user defined key that is input
--It takes in the key then populates the four words with there repected values in their positions. If
```

```
--load is one it signifies that it is round zero and it may begin the process when there is a rising edge
--in the clock pulse
	KEYCreator: process(CLOCK)
	begin
		if rising_edge(CLOCK) then
			if load = '1' then
--if its the first round then populate the variable word with the user defined key.
				Word32(0) <= KEY_IN(127 downto 96);
				Word32(1) <= KEY_IN(95 downto 64);
				Word32(2) <= KEY_IN(63 downto 32);
				Word32(3) <= KEY_IN(31 downto 0);
--else if its any other round less than 10 populate word with the new value caculated
			elsif RoundNum < (x"a" & "00")
				then
				Word32 <= Word32(1 to 3) & (Word32(0) xor NewWord);
			end if;
		end if;
	end process;
--Routine word is a one byte left circular shift
	RoutineWord <=		Word32(3)(23 downto 0) & Word32(3)(31 downto 24);
	NewBytesFORKey: for k in 0 to 3 generate
--instead of using  seperate module called sbox creating an array/look up table(LUT) populating it accordingly
--makes for easier coding. The sbox array is filled with the specified values from the AES look up table.
		type sboxKey is array(0 to 255) of std_logic_vector(7 downto 0);
		constant KeySub : sboxKey :=(
			x"63", x"7c", x"77", x"7b", x"f2", x"6b", x"6f", x"c5", x"30", x"01", x"67", x"2b", x"fe", x"d7", x"ab", x"76",
			x"ca", x"82", x"c9", x"7d", x"fa", x"59", x"47", x"f0", x"ad", x"d4", x"a2", x"af", x"9c", x"a4", x"72", x"c0",
			x"b7", x"fd", x"93", x"26", x"36", x"3f", x"f7", x"cc", x"34", x"a5", x"e5", x"f1", x"71", x"d8", x"31", x"15",
```

```
x"04", x"c7", x"23", x"c3", x"18", x"96", x"05", x"9a", x"07", x"12", x"80", x"e2", x"eb", x"27", x"b2", x"75",
x"09", x"83", x"2c", x"1a", x"1b", x"6e", x"5a", x"a0", x"52", x"3b", x"d6", x"b3", x"29", x"e3", x"2f", x"84",
x"53", x"d1", x"00", x"ed", x"20", x"fc", x"b1", x"5b", x"6a", x"cb", x"be", x"39", x"4a", x"4c", x"58", x"cf",
x"d0", x"ef", x"aa", x"fb", x"43", x"4d", x"33", x"85", x"45", x"f9", x"02", x"7f", x"50", x"3c", x"9f", x"a8",
x"51", x"a3", x"40", x"8f", x"92", x"9d", x"38", x"f5", x"bc", x"b6", x"da", x"21", x"10", x"ff", x"f3", x"d2",
x"cd", x"0c", x"13", x"ec", x"5f", x"97", x"44", x"17", x"c4", x"a7", x"7e", x"3d", x"64", x"5d", x"19", x"73",
x"60", x"81", x"4f", x"dc", x"22", x"2a", x"90", x"88", x"46", x"ee", x"b8", x"14", x"de", x"5e", x"0b", x"db",
x"e0", x"32", x"3a", x"0a", x"49", x"06", x"24", x"5c", x"c2", x"d3", x"ac", x"62", x"91", x"95", x"e4", x"79",
x"e7", x"c8", x"37", x"6d", x"8d", x"d5", x"4e", x"a9", x"6c", x"56", x"f4", x"ea", x"65", x"7a", x"ae", x"08",
x"ba", x"78", x"25", x"2e", x"1c", x"a6", x"b4", x"c6", x"e8", x"dd", x"74", x"1f", x"4b", x"bd", x"8b", x"8a",
x"70", x"3e", x"b5", x"66", x"48", x"03", x"f6", x"0e", x"61", x"35", x"57", x"b9", x"86", x"c1", x"1d", x"9e",
x"e1", x"f8", x"98", x"11", x"69", x"d9", x"8e", x"94", x"9b", x"1e", x"87", x"e9", x"ce", x"55", x"28", x"df",
x"8c", x"a1", x"89", x"0d", x"bf", x"e6", x"42", x"68", x"41", x"99", x"2d", x"0f", x"b0", x"54", x"bb", x"16");

begin

--The subroutineword variable is filled with the new values derived from the sbox array

SubRoutineWord(8*(k+1)-1 downto 8*k) <= KeySub(conv_integer(RoutineWord(8*(k+1)-1 downto 8*k)));

end generate;

--The subroutine word is the XORed with the round constant value corresponding to the ongoing round. These round constants are uses

-- to get rid of any simularities created by the exspansion process. There must be zeros for 3 of the least significant bits.

NewWord <=	SubRoutineWord xor (RoundConstantV(conv_integer(RoundNum(5 downto 2))) & x"000000") when RoundNum(1 downto 0) = "00"
```

```
                    else
--else it is XORed with the populated word32(3) as can be seen above the lut.
                              Word32(3);
--The new key value is created which is the 4 words concantanated together to create the 128-bit output for the next module
          NewKey <= Word32(0) & Word32(1) & Word32(2) & Word32(3);


--creating the constant array Submemory makes for easier codeing.
--It eliminates the need for the S_box module, this module takes in the input
--from the key expansion module. Then Each input byte is replaced by a sub byte
--after it is put through the S-box. Doing this stops linearity occurring in the
--cipher. These S-boxes were designed using the multiplicated inverse over the
--Galois field 2^8 which is known for its non-linearity.
          ByteSubstitutionFOR: for k in 0 to 15 generate
                    type sboxSub is array(0 to 255) of std_logic_vector(7 downto 0);
                    constant subRam : sboxSub := (
--The input into the s-box is 8 bits and so is the output. This is done 16
--times which means there are 16 S-boxes and they are all running is parallel.
                              x"63", x"7c", x"77", x"7b", x"f2", x"6b", x"6f", x"c5", x"30", x"01", x"67", x"2b", x"fe", x"d7", x"ab", x"76",
                              x"ca", x"82", x"c9", x"7d", x"fa", x"59", x"47", x"f0", x"ad", x"d4", x"a2", x"af", x"9c", x"a4", x"72", x"c0",
                              x"b7", x"fd", x"93", x"26", x"36", x"3f", x"f7", x"cc", x"34", x"a5", x"e5", x"f1", x"71", x"d8", x"31", x"15",
                              x"04", x"c7", x"23", x"c3", x"18", x"96", x"05", x"9a", x"07", x"12", x"80", x"e2", x"eb", x"27", x"b2", x"75",
                              x"09", x"83", x"2c", x"1a", x"1b", x"6e", x"5a", x"a0", x"52", x"3b", x"d6", x"b3", x"29", x"e3", x"2f", x"84",
                              x"53", x"d1", x"00", x"ed", x"20", x"fc", x"b1", x"5b", x"6a", x"cb", x"be", x"39", x"4a", x"4c", x"58", x"cf",
                              x"d0", x"ef", x"aa", x"fb", x"43", x"4d", x"33", x"85", x"45", x"f9", x"02", x"7f", x"50", x"3c", x"9f", x"a8",
                              x"51", x"a3", x"40", x"8f", x"92", x"9d", x"38", x"f5", x"bc", x"b6", x"da", x"21", x"10", x"ff", x"f3", x"d2",
```

```
                    x"cd", x"0c", x"13", x"ec", x"5f", x"97", x"44", x"17", x"c4", x"a7", x"7e",
x"3d", x"64", x"5d", x"19", x"73",
                    x"60", x"81", x"4f", x"dc", x"22", x"2a", x"90", x"88", x"46", x"ee", x"b8",
x"14", x"de", x"5e", x"0b", x"db",
                    x"e0", x"32", x"3a", x"0a", x"49", x"06", x"24", x"5c", x"c2", x"d3", x"ac",
x"62", x"91", x"95", x"e4", x"79",
                    x"e7", x"c8", x"37", x"6d", x"8d", x"d5", x"4e", x"a9", x"6c", x"56", x"f4",
x"ea", x"65", x"7a", x"ae", x"08",
                    x"ba", x"78", x"25", x"2e", x"1c", x"a6", x"b4", x"c6", x"e8", x"dd", x"74",
x"1f", x"4b", x"bd", x"8b", x"8a",
                    x"70", x"3e", x"b5", x"66", x"48", x"03", x"f6", x"0e", x"61", x"35", x"57",
x"b9", x"86", x"c1", x"1d", x"9e",
                    x"e1", x"f8", x"98", x"11", x"69", x"d9", x"8e", x"94", x"9b", x"1e", x"87",
x"e9", x"ce", x"55", x"28", x"df",
                    x"8c", x"a1", x"89", x"0d", x"bf", x"e6", x"42", x"68", x"41", x"99", x"2d",
x"0f", x"b0", x"54", x"bb", x"16");
    begin
        process(CLOCK)
        begin
            if rising_edge(CLOCK) then
                if RoundNum(1 downto 0) = "00" and RoundNum < (x"a" & "00")
then
--As it can be seen the ADDkey values are input and then converted to the repected values to find the new ones in
--the sbox array. The new values then populate the output of the module BYTRSUB
                    BYTESUB(8*(k+1)-1 downto 8*k) <=
subRam(conv_integer(ADDkey(8*(k+1)-1 downto 8*k)));
                end if;
            end if;
        end process;
    end generate;


--This module differs to the mixedcolumn module as can be seen. There was an issue while testing when combining the modues
--into one. As it can be seen it loops 16 times with byte sized values rather than looping 4 times with word sized values.
```

```
--When this change was made the algorithm came together and worked correctly. The input i changed to matrix form then
--rows are shifted as seen below.
 Vector_To_Matrix: PROCESS(clock)
   BEGIN
--If statment making sure both parameters are 1 before changes are made.
      if rising_edge(clock) then
         FOR k IN 15 downto 0 LOOP--for loop mapping the new values to the Input_Mtrix
                  Input_Matrix(15-k) <= BYTESUB(8*k+7 downto 8*k);
         END LOOP;--Values are stored here temporarily end loop
      end if;--end if stament
   END PROCESS Vector_To_Matrix;--function is complete


--Map the newly shifted matrix to vector form so it can be fed onto the next module
   Shifted_Matrix_to_Vector: PROCESS(Shifted_Matrix)
   BEGIN--start of function
      if rising_edge(clock) then
         FOR k IN 15 downto 0 LOOP--for loop mapping the newly shifted matrix values
                                                      RowShift(8*k+7 DOWNTO 8*k) <= Shifted_Matrix(15-k);--
converting back to vector form
         END LOOP;
      end if;--end if
   END PROCESS Shifted_Matrix_to_Vector;-- The process/function is complete


--Once the values are converted to matrix form
--the values are shifted accordingly, the first
--row is shift 0 places, the second row shifted
--one place to the left, the third 2 places and
--the last row is shift 4 places to the left.
-----------------------------------------------------------------
-----------------------------------------------------------------
   Shifted_Matrix(0)  <=  Input_Matrix(0);--no shift
```

```
   Shifted_Matrix(1)  <=  Input_Matrix(5);--shifted one place to the left
   Shifted_Matrix(2)  <=  Input_Matrix(10);--shifted two places to the left
   Shifted_Matrix(3)  <=  Input_Matrix(15);--shifted three places to the left
--         ------------------------------------------------------------
   Shifted_Matrix(4)  <=  Input_Matrix(4);--no shift
   Shifted_Matrix(5)  <=  Input_Matrix(9);--shifted one place to the left
   Shifted_Matrix(6)  <=  Input_Matrix(14);--shifted two places to the left
   Shifted_Matrix(7)  <=  Input_Matrix(3);--shifted three places to the left
-----------------------------------------------------------------
   Shifted_Matrix(8)  <=  Input_Matrix(8);--no shift
   Shifted_Matrix(9)  <=  Input_Matrix(13);--shifted one place to the left
   Shifted_Matrix(10) <=  Input_Matrix(2);--shifted two places to the left
   Shifted_Matrix(11) <=  Input_Matrix(7);--shifted three places to the left
-----------------------------------------------------------------
   Shifted_Matrix(12) <=  Input_Matrix(12);--no shift
   Shifted_Matrix(13) <=  Input_Matrix(1);--shifted one place to the left
   Shifted_Matrix(14) <=  Input_Matrix(6);--shifted two places to the left
   Shifted_Matrix(15) <=  Input_Matrix(11);--shifted three places to the left


--This module will be used in 9 of the rounds but wont called on the 10th round. This
--module takes an input in vector form converts in to matrix to apply the nessecary mathematics.
--which is a multiply by one, two and three. Then XORing the outputs of these multiplications
--with the respected correspondent to populate the output matrix. Which is then converted back
--to a vector so it can be fed on to the next stage of the algorithm.
        MIXEDCOLUMNS : for k in 0 to 3 generate
--Declaring signals specific to this module.
                signal InputMatrix, OutPutMatrix : std_logic_vector(31 downto 0);
                signal BY1i, BY1j, BY1k, BY1l, MATRIXOUT1, MATRIXOUT2, MATRIXOUT3,
MATRIXOUT4 : std_logic_vector(7 downto 0);
                signal BY2i, BY2J, BY2k, BY2l, BY3i, BY3j, BY3k, BY3l : std_logic_vector(7 downto 0);
```

```
        begin
--Mapping input values to a local signal creating matrix
                InputMatrix <= RowShift(32*(k+1)-1 downto 32*k);
-----------------------------------------------------------------------------------------------
                BY1i <= InputMatrix(31 downto 24);
                BY1j <= InputMatrix(23 downto 16);
                BY1k <= InputMatrix(15 downto 8);
                BY1l <= InputMatrix(7 downto 0);
--Multiplying by 2 over the galois field.This is done so that if there is any values exceeding the seven bit field
--it will be XORed with the vector which it cant be deduced. 1B in HEX, 00011011 in Binary or 28 in decimel.
-----------------------------------------------------------------------------------------------
                BY2i <= BY1i(6 downto 0) & '0' when BY1i(7) = '0' else (BY1i(6 downto 0) & '0') xor x"1b";
                BY2J <= BY1j(6 downto 0) & '0' when BY1j(7) = '0' else (BY1j(6 downto 0) & '0') xor x"1b";
                BY2k <= BY1k(6 downto 0) & '0' when BY1k(7) = '0' else (BY1k(6 downto 0) & '0') xor x"1b";
                BY2l <= BY1l(6 downto 0) & '0' when BY1l(7) = '0' else (BY1l(6 downto 0) & '0') xor x"1b";
--Multiplication by 3 is done by simply XORing the multiplied by two and the multiplied by
-----------------------------------------------------------------------------------------------
                BY3i <= BY1i xor BY2i;
                BY3j <= BY1j xor BY2J;
                BY3k <= BY1k xor BY2k;
                BY3l <= BY1l xor BY2l;
--Addtion of the galois fields using the bitwise function XOR
-----------------------------------------------------------------------------------------------
                MATRIXOUT1 <= BY2i xor BY3j xor BY1k xor BY1l;
                MATRIXOUT2 <= BY1i xor BY2j xor BY3k xor BY1l;
                MATRIXOUT3 <= BY1i xor BY1j xor BY2k xor BY3l;
                MATRIXOUT4 <= BY3i xor BY1j xor BY1k xor BY2l;
```

```
-----------------------------------------------------------------------------------------------
--mapping the values taht are taking from the calculations
                    OutPutMatrix <= MATRIXOUT1 & MATRIXOUT2 & MATRIXOUT3 & MATRIXOUT4;
--Mapping back to vector to be fed to next round unles its round 9
                    MixedColumnOut(32*(k+1)-1 downto 32*k) <= OutPutMatrix;
          end generate;
----------------------------------------------------------------------------------------------
--The variable NewRound will equal rowshift only when it is the last round other than that
          NewRound <=   RowShift when RoundNum(5 downto 2) = x"a"
                    else
                              MixedColumnOut;
--For the first round the plaintext is XORed with the Key created from the key creator module
--After the first round it is XORed with the output from the MixCol this is because that is the
--end of the round. For the final round the Mix col module is dropped and it will be the row shifting
--module taht the key will be XORed with.
          ADDkey <=         D_IN xor KEY_IN when RoundNum = "000000"
                    else
                               NewRound xor NewKey;
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--The cipher text that is ouput from each round
          D_OUT <= ADDkey;
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
--State machine that will count the rounds with a reset process that brings the counter
--back to round 000000.
          ControllingStateMachine: process(CLOCK)
          begin
                    if rising_edge(CLOCK) then
```

```
--sets the round value to 11 so it will reset the counter.
                    if RESET = '1' then
                        RoundNum <= x"a" & "01";
                    else
--if it is round eleven then reset the round counter.
                        if RoundNum = (x"a" & "01") then
                            if LOAD = '1' then
                                RoundNum <= "000000";
                            end if;
--if the round number is not equal 11 then increment the counter
                        elsif RoundNum /= (x"a" & "01") then
                            RoundNum <= RoundNum + '1';
                        end if;
                    end if;
                end if;
        end process;
--Good shows when the 10th round has been iterated by the value 1 else it shows a 0.
        GOOD <= '1' when RoundNum = x"a" & "00" else '0';
--Complete shows the value 1 after the 10th round is complete
        Complete <= '1' when RoundNum = x"a" & "01" else '0';


        end Behavioral;
```