```
--
-- This entity provides a rudimentary SCC function.  This block generates and uses baud
-- rates determined by the equation baudval = (external clock freq. / Baud * 16).
-- Data is transmitted LSB first. The modes of operation are: 8 data bits with even or odd
 parity,
-- one start bit and one stop bit or 7 data bits with even or odd parity, one start bit an
d one
-- stop bit. Four active-high direct status signals some of which can be used for interrup
ts: parity error,
-- overflow, txrdy, and receive_full are generated. The configuration of the SCC's baud ra
te is done
-- at compile time see "COMPILETIME CONFIGERATION SELECTIONS" section in the Clock_gen.vhd
 file. The receive
-- data and transmit data are double-buffered.
--
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library synplify;
use synplify.attributes.all;

entity SCC is port (
    reset_n         : in        std_logic;
    clk             : in        std_logic;
    WEn             : in        std_logic;
    OEn             : in        std_logic;
--    SCC_csn         : in        std_logic;
    SCC_data_in     : in        std_logic_vector(7 downto 0);
    SCCrx           : in        std_logic;
--    baud_val        : in        std_logic_vector(7 downto 0);
    -- Configuration bits
--    bit8            : in        std_logic; -- if set to one 8 data bits otherwise 7 data
 bits
--    parity_en       : in        std_logic; -- if set to one parity is enabled otherwise
disabled
--    odd_n_even      : in        std_logic; -- if set to one odd parity otherwise even pa
rity

    -- Status bits
    parity_err      : out       std_logic; -- parity error indicator on recieved data
    overflow        : out       std_logic; -- receiver overflow
    txrdy           : out       std_logic; -- transmit ready for another byte
    receive_full    : out       std_logic; -- receiver has a byte ready
    SCC_data_out    : out       std_logic_vector(7 downto 0);
    SCCtx           : out       std_logic
);
end SCC;

architecture rtl of SCC is
attribute syn_radhardlevel of rtl : architecture is "tmr";

------------------------------------------------------------
-- COMPONENT DECLARATIONS
------------------------------------------------------------

    component Tx_async port (
       clk                    : in  std_logic;                      -- system clock
       xmit_pulse             : in  std_logic;                      -- transmit pulse
       reset_n                : in  std_logic;                      -- active low async res
et
       rst_tx_empty           : in  std_logic;                      -- reset transmit empty
       tx_hold_reg            : in  std_logic_vector(7 downto 0);   -- transmit byte hold r
egister
       bit8                   : in  std_logic;                      -- if set to one 8 data
 bits otherwise 7 data bits
```

```
       parity_en              : in  std_logic;                     -- if set to one parity
 is enabled otherwise disabled
       odd_n_even             : in  std_logic;                     -- if set to one odd pa
rity otherwise even parity

       txrdy                  : out std_logic;                     -- transmit ready for a
nother byte
       SCCtx                  : out std_logic                      -- serial data stream o
ut
      );
   end component;

   component Rx_async port (
       clk                   : in   std_logic;                    -- system clock
       baud_clock            : in   std_logic;                    -- 8x baud clock pulse
       reset_n               : in   std_logic;                    -- active low async res
et
       bit8                  : in   std_logic;                    -- if set to one 8 data
 bits otherwise 7 data bits
       parity_en             : in   std_logic;                    -- if set to one parity
 is enabled otherwise disabled
       odd_n_even            : in   std_logic;                    -- if set to one odd pa
rity otherwise even parity
       read_rx_byte          : in   std_logic;                    -- read rx byte registe
r
       SCCrx                 : in   std_logic;

       overflow              : out  std_logic;                    -- receiver overflow
       parity_err            : out  std_logic;                    -- parity error indicat
or on recieved data
       receive_full          : out  std_logic;                    -- receiver has a byte
ready
       rx_byte               : out  std_logic_vector(7 downto 0)   -- receive byte registe
r
      );
   end component;


   component Clock_gen port (
             clk              : in   std_logic;                     -- system clock
             reset_n          : in   std_logic;                     -- active low async re
set
             baud_val         : in   std_logic_vector(4 downto 0);  -- value loaded into c
ntr

             baud_clock       : out  std_logic;                     -- 8x baud clock pulse
             xmit_pulse       : out  std_logic                      -- transmit pulse
            );
   end component;

-------------------------------------------------------
-- INTERNAL SIGNAL DECLARATIONS
-------------------------------------------------------

    signal xmit_pulse       : std_logic;                      -- transmit pulse
    signal baud_clock       : std_logic;                      -- 8x baud clock pulse
    signal rst_tx_empty     : std_logic;                      -- reset transmit empty
    signal tx_hold_reg      : std_logic_vector(7 downto 0);   -- transmit byte hold registe
r
    signal read_rx_byte     : std_logic;                      -- read rx byte register
    signal rx_byte          : std_logic_vector(7 downto 0);   -- receive byte register

-------------------------------------------------------
-- MODE CONSTANT, 1 for synchronous 0 for asynchronous
-------------------------------------------------------
--    constant SYNCHRONOUS  : std_logic := '0';
    constant baud_val     : std_logic_vector(7 downto 0) := "00010101"; -- 21 for 40, 18 f
or 33.5 -- 1 for simulation
```

```
    constant bit8         : std_logic := '1';
    constant parity_en    : std_logic := '1';
    constant odd_n_even   : std_logic := '0';
    constant SCC_csn      : std_logic := '0';

begin
    -- hookup lower levels
UG01: Clock_gen port map (
                          clk        => clk,
                          reset_n    => reset_n,
                          baud_val  => baud_val(4 downto 0),
                          baud_clock => baud_clock,
                          xmit_pulse => xmit_pulse
                         );


UG02: Tx_async port map (
                         clk          => clk,
                         xmit_pulse   => xmit_pulse,
                         reset_n      => reset_n,
                         rst_tx_empty => rst_tx_empty,
                         tx_hold_reg  => tx_hold_reg,
                         bit8         => bit8,
                         parity_en    => parity_en,
                         odd_n_even   => odd_n_even,
                         txrdy        => txrdy,
                         SCCtx        => SCCtx
                        );

UG04: Rx_async port map (
                         clk          => clk,
                         baud_clock   => baud_clock,
                         reset_n      => reset_n,
                         bit8         => bit8,
                         parity_en    => parity_en,
                         odd_n_even   => odd_n_even,
                         read_rx_byte => read_rx_byte,
                         SCCrx        => SCCrx,
                         overflow     => overflow,
                         parity_err   => parity_err,
                         receive_full => receive_full,
                         rx_byte      => rx_byte
                        );

-------------------------------------------------------------------------------
-- cpu writes to SCC registers
-------------------------------------------------------------------------------
reg_write : process(clk, reset_n)
begin
if(reset_n = '0') then
    tx_hold_reg <= (others => '0');
elsif(clk'event and clk = '1') then
    if(SCC_csn = '0' and WEn = '0') then
      tx_hold_reg <= SCC_data_in;
    end if;
end if;
end process;

rst_tx_empty <= '1' when WEn = '0' and SCC_csn = '0' else '0';

-------------------------------------------------------------------------------
-- cpu reads from SCC registers
-------------------------------------------------------------------------------

SCC_data_out <= rx_byte when OEn = '0' and SCC_csn = '0' else (others => '0');

read_rx_byte <= '1' when SCC_csn = '0' and OEn = '0' else '0';
```

```
END rtl;
```