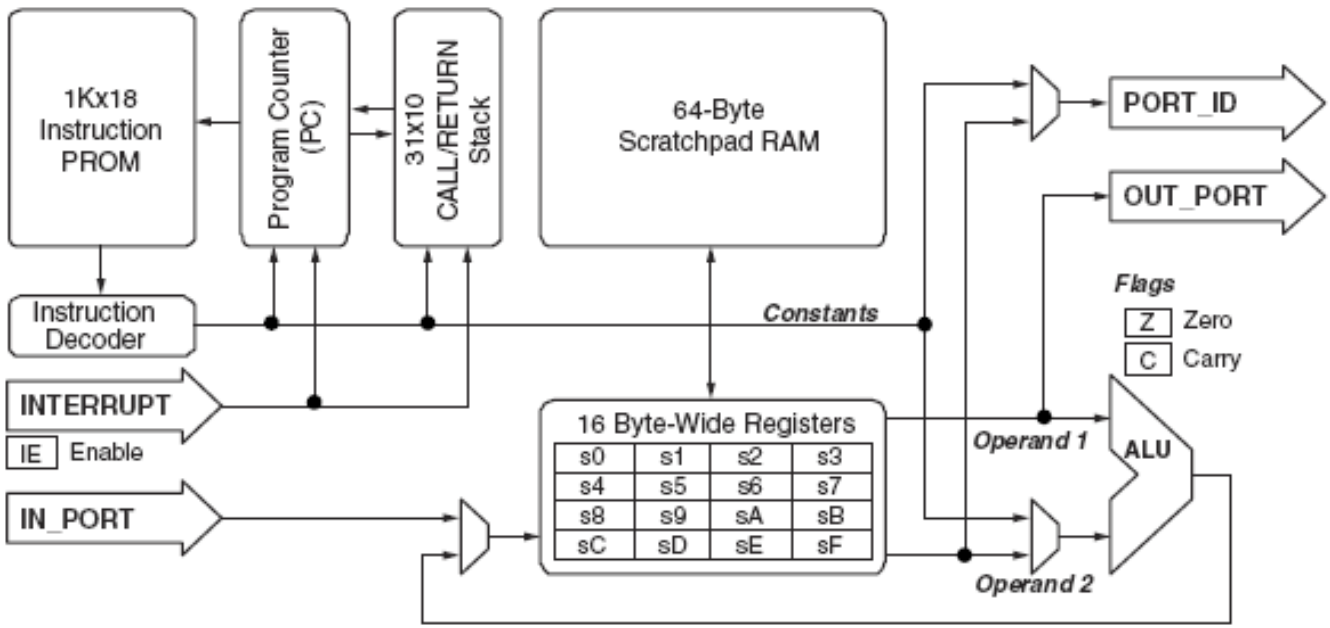
EE3810 – Lecture 21

Using Soft Core Processors: The PicoBlaze RISC Microcontroller

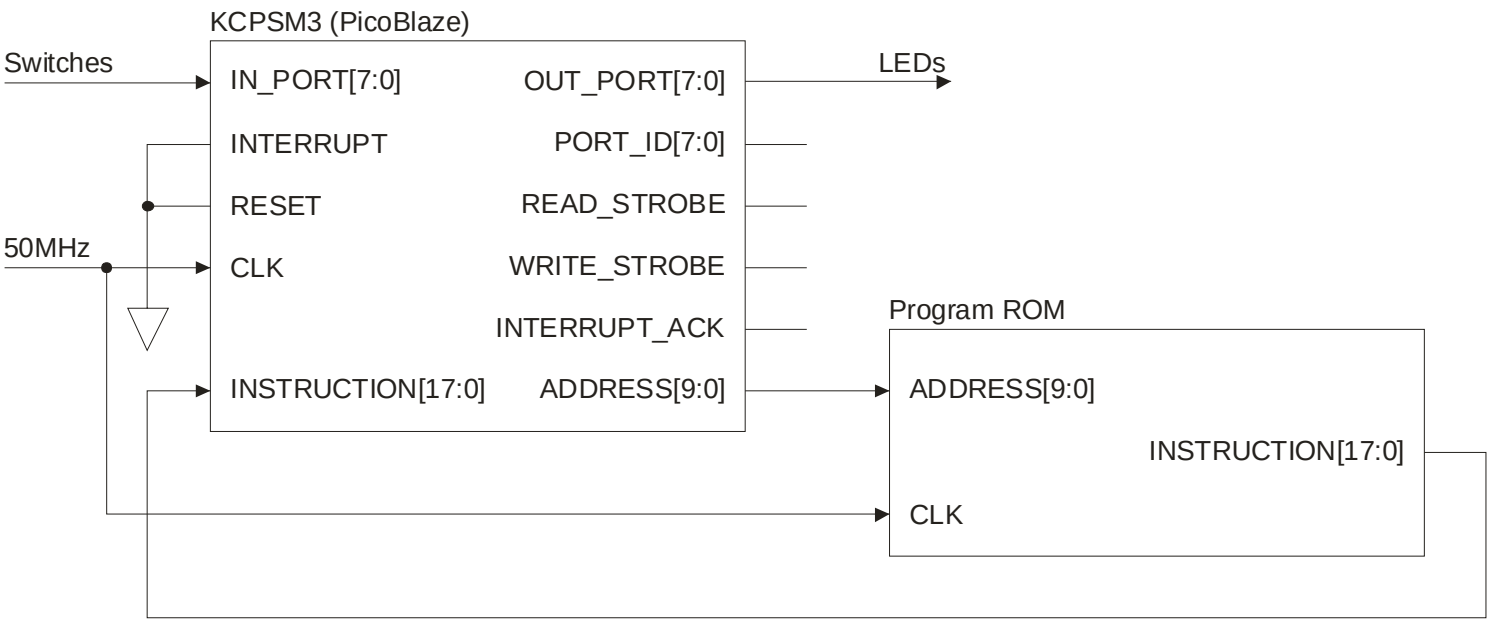
What is the PicoBlaze?

- The PicoBlaze is an IP Core which provides a simple, 8-bit programmable microcontroller:



A KISS Example

- Consider the following (pretty minimal) PicoBlaze-based system:



Getting the PicoBlaze

- Step 1: download the PicoBlaze files from the Xilinx website.
  - You'll need to register to get access to the PicoBlaze lounge.
  - Download the Users Guide (UG129)
  - Download the Spartan-3 version of the processor
- http://www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm

Instantiating a PicoBlaze

- The PicoBlaze is provided as a set of VHDL files.
  - One component, KCPSM3, is the PicoBlaze
  - The other, prog_rom, is a program ROM template.
- Other files provide an assembler and additional documentation. It’s all good…

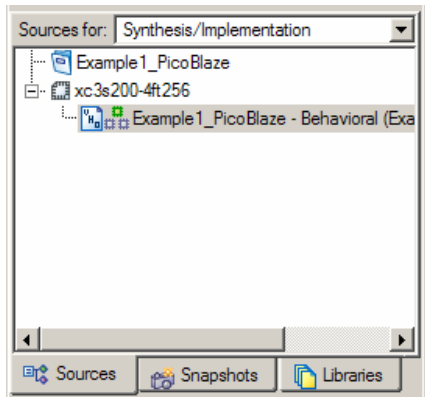
Creating The VHDL

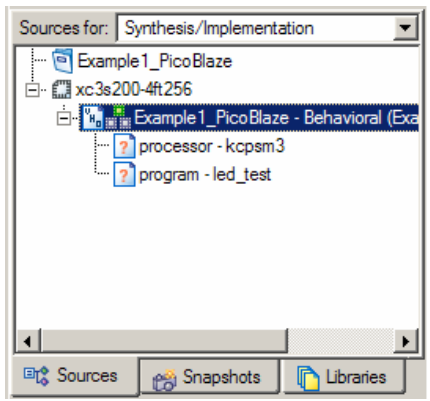
```
entity Example1_PicoBlaze is
    Port ( clk_50MHz : in  std_logic;
           switches  : in  std_logic_vector (7 downto 0);
           leds      : out std_logic_vector (7 downto 0));
end Example1_PicoBlaze;

architecture Behavioral of Example1_PicoBlaze is
   component KCPSM3
      port  (
         address      : out std_logic_vector( 9 downto 0);
         instruction  : in  std_logic_vector(17 downto 0);
         port_id      : out std_logic_vector( 7 downto 0);
         write_strobe : out std_logic;
         out_port     : out std_logic_vector( 7 downto 0);
         read_strobe  : out std_logic;
         in_port      : in  std_logic_vector( 7 downto 0);
         interrupt    : in  std_logic;
         interrupt_ack : out std_logic;
         reset         : in  std_logic;
         clk           : in  std_logic
            );
   end component;

   component led_test
      port (
         address     : in  std_logic_vector( 9 downto 0);
         instruction : out std_logic_vector(17 downto 0);
         clk         : in  std_logic
            );
   end component;
```

```
   signal addx_bus : std_logic_vector(9 downto 0);
   signal inst_bus : std_logic_vector(17 downto 0);
begin
   processor: kcpsm3
      port map(
         address       => addx_bus,
         instruction   => inst_bus,
         port_id       => open,
         write_strobe  => open,
         out_port      => leds,
         read_strobe   => open,
         in_port       => switches,
         interrupt     => '0',
         interrupt_ack => open,
         reset         => '0',
         clk           => clk_50MHz
            );

   program: led_test
      port map(
         address     => addx_bus,
         instruction => inst_bus,
         clk         => clk_50MHz
            );
end Behavioral;
```

ISE Behavior

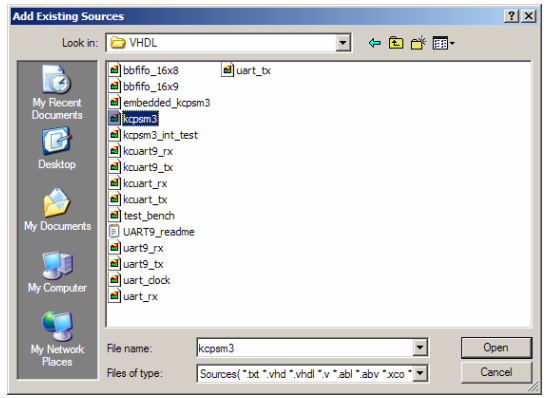
- Before saving, just the model being developed is shown.
- After saving, we see placeholders for the components:
  - PicoBlaze
  - Program ROM

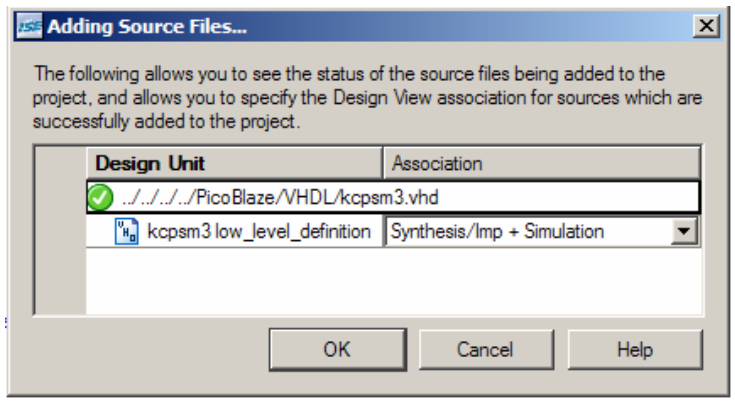




Adding the KCPSM3 Source

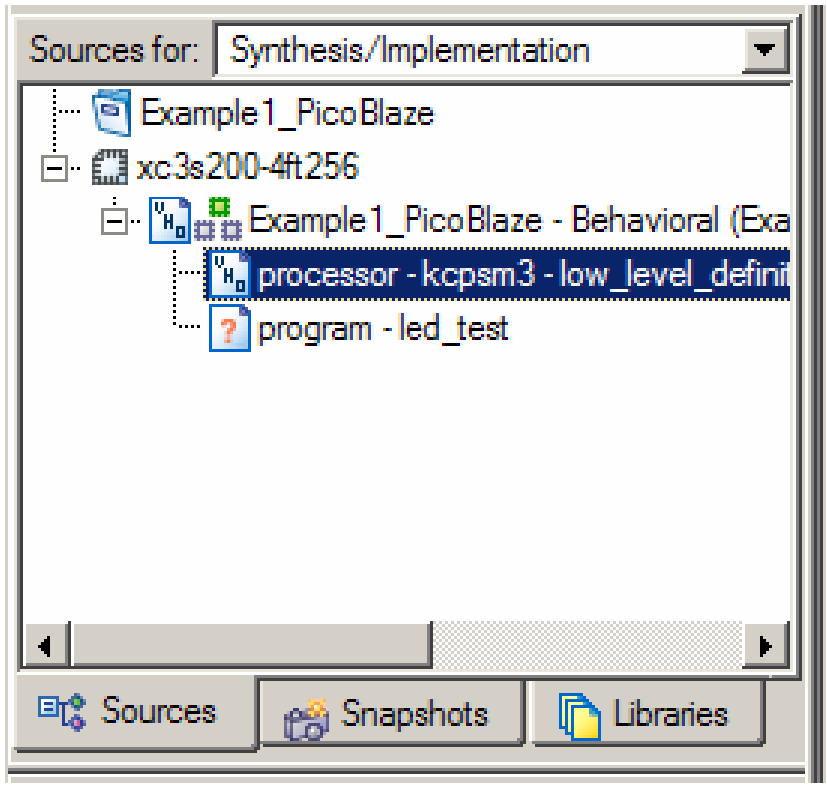
- Off the Project menu, choose “Add Source.”
- Browse to where you have the KCPSM3 VHDL file located.
- Open the file.
- Click OK.



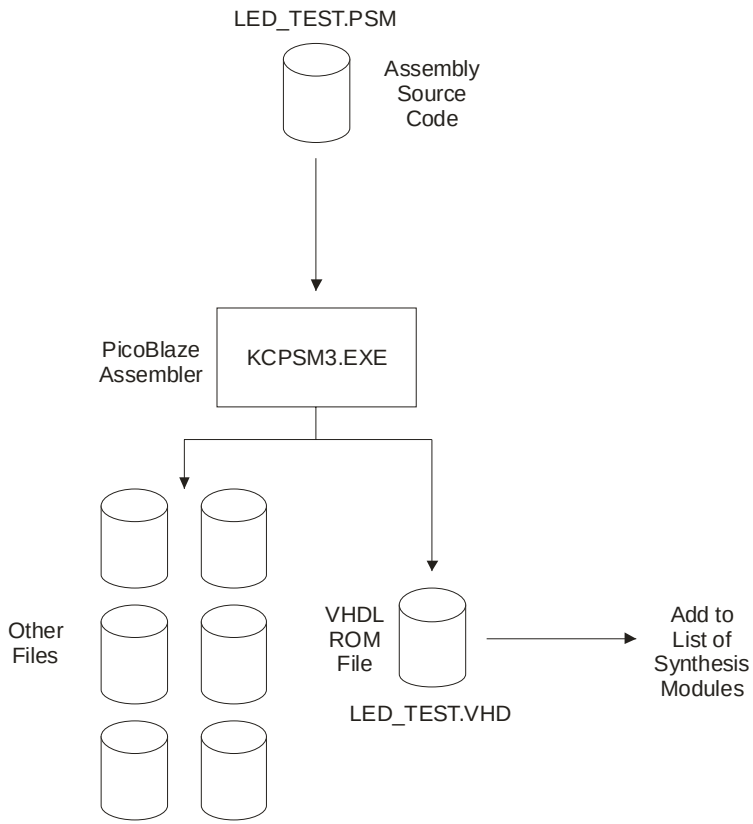


Almost Done

- We've now added the PicoBlaze source files to the project.
- Next, we need to write and compile the assembly code that the PicoBlaze will execute.



Compiler Flow



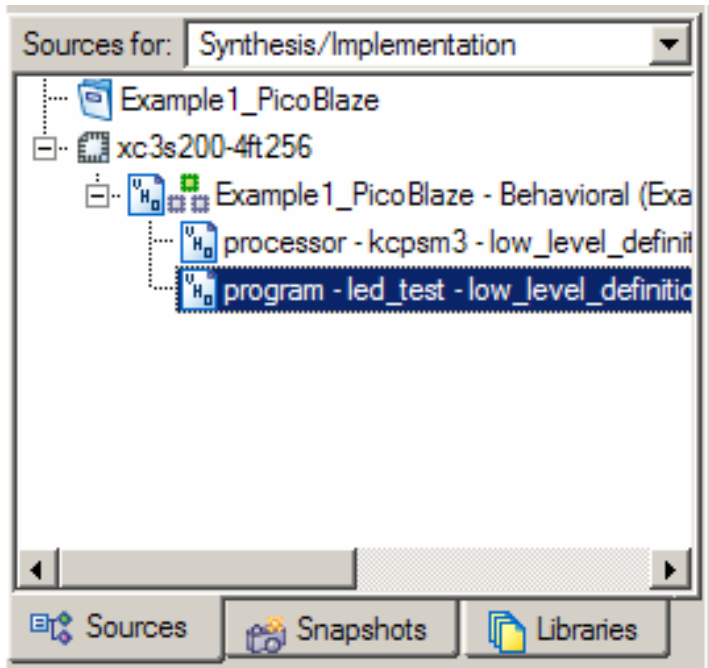
Example Source Code

- For this example we just need to read the switches and pass the data to the LEDs.
- All addresses will behave the same way – I just picked one to use.

```
; Pass switch settings to LEDs.
;
; For this routine, all I/O addresses
; produce the same result.
;
start: INPUT  s0, 00  ;read switches.
       OUTPUT s0, 00  ;write leds.
       JUMP   start
```

Adding the ROM Module

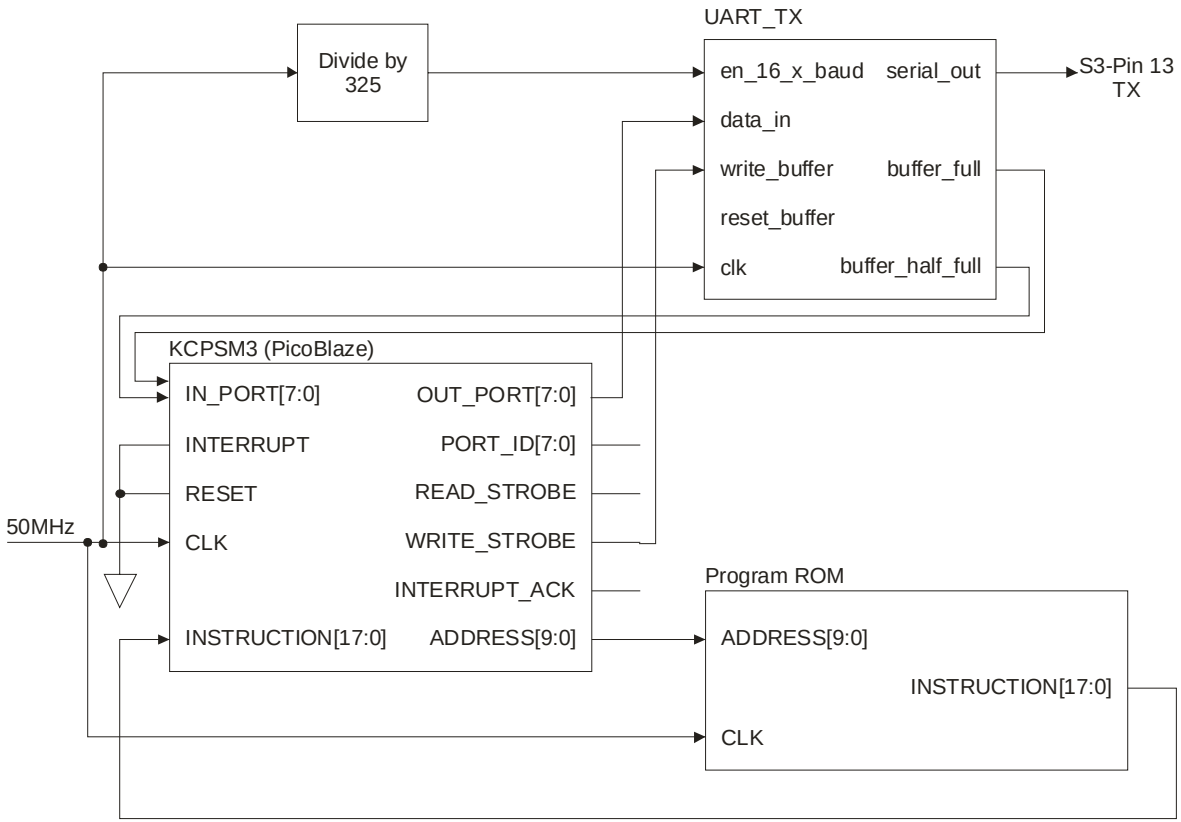
- As before, we “add file” by browsing to the location of the assembler output and adding the file.
- Be sure to add the .vhd file, not the Verilog file!
- Now we can synthesize the design and see if it works!



Adding a UART

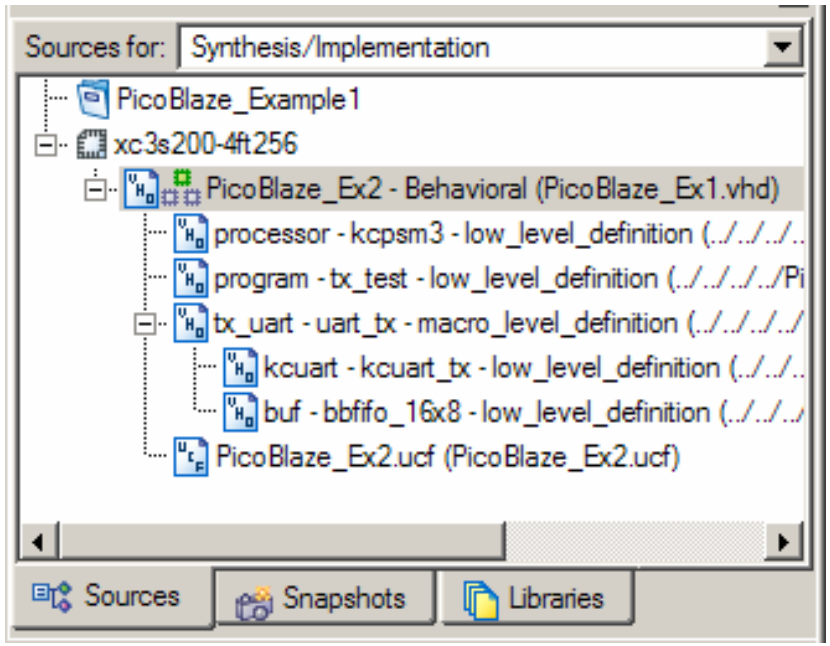
- Similarly, we can add a simple UART to our design to make a system that can communicate with a PC using RS-232.
- For this example, we'll just include the transmit half of the UART.
- Adding the receive side of the UART is done in a similar way.

System Schematic



Adding the UART Modules

- The UART transmitter contains three modules:
  - tx_uart
  - kcuart
  - bbfifo_16x8
- Add these modules like we did with the PicoBlaze.



The UART VHDL

```
entity PicoBlaze_Ex2 is
    Port ( clk : in  std_logic;
           txd : out std_logic);
end PicoBlaze_Ex2;

architecture Behavioral of PicoBlaze_Ex2 is
   component KCPSM3
      port  (
         address      : out std_logic_vector( 9 downto 0);
         instruction  : in  std_logic_vector(17 downto 0);
         port_id      : out std_logic_vector( 7 downto 0);
         write_strobe : out std_logic;
         out_port     : out std_logic_vector( 7 downto 0);
         read_strobe  : out std_logic;
         in_port      : in  std_logic_vector( 7 downto 0);
         interrupt    : in  std_logic;
         interrupt_ack : out std_logic;
         reset        : in  std_logic;
         clk          : in  std_logic
            );
   end component;
```

The UART VHDL, continued

```
-- PicoBlaze ROM File
   component tx_test
      port (
         address     : in  std_logic_vector( 9 downto 0);
         instruction : out std_logic_vector(17 downto 0);
         clk         : in  std_logic
           );
   end component;

-- UART Module
   component uart_tx
      port (
         data_in          : in  std_logic_vector(7 downto 0);
         write_buffer     : in  std_logic;
         reset_buffer     : in  std_logic;
         en_16_x_baud     : in  std_logic;
         serial_out       : out std_logic;
         buffer_full      : out std_logic;
         buffer_half_full : out std_logic;
         clk              : in  std_logic
           );
   end component;
```

The UART VHDL, continued

```
-- signals for processor
   signal address_signal : std_logic_vector(9 downto 0);
   signal instruction_signal : std_logic_vector(17 downto 0);
   signal pb_inp_data : std_logic_vector(7 downto 0);
   signal pb_out_data : std_logic_vector(7 downto 0);
-- signals for interfacing uart
   signal baud_count : integer range 0 to 511 := 0;
   signal en_16_x_baud : std_logic;
   signal uart_write : std_logic;
begin
   leds <= switches;
   processor: kcpsm3
      port map(
         address => address_signal,
         instruction => instruction_signal,
         port_id => open,
         write_strobe => uart_write,
         out_port => pb_out_data,
         read_strobe => open,
         in_port => pb_inp_data,
         interrupt => '0',
         interrupt_ack => open,
         reset => '0',
         clk => clk);
```

The UART VHDL, continued

```
program: tx_test
   port map(
      address     => address_signal,
      instruction => instruction_signal,
      clk         => clk
           );

tx_uart: uart_tx
   port map (
      data_in          => pb_out_data,
      write_buffer     => uart_write,
      reset_buffer     => '0',
      en_16_x_baud     => en_16_x_baud,
      serial_out       => txd,
      buffer_full      => pb_inp_data(7),
      buffer_half_full => pb_inp_data(0),
      clk              => clk
            );
```

```
baud_timer: process ( clk )
begin
   if clk'event and clk = '1' then
      if baud_count = 325 then
         baud_count   <= 0;
         en_16_x_baud <= '1';
      else
         baud_count   <= baud_count + 1;
         en_16_x_baud <= '0';
      end if;
   end if;
end process baud_timer;
end Behavioral;
```

UART Transmit Code

```
; UART Transmit Example
;
start:    LOAD  s1, 45        ;ASCII "E".
          CALL  xmit          ;Send character.
          LOAD  s1, 45        ;ASCII "E".
          CALL  xmit          ;Send character.
          LOAD  s1, 33        ;ASCII "3".
          CALL  xmit          ;Send character.
          LOAD  s1, 38        ;ASCII "8".
          CALL  xmit          ;Send character.
          LOAD  s1, 31        ;ASCII "1".
          CALL  xmit          ;Send character.
          LOAD  s1, 30        ;ASCII "0".
          CALL  xmit          ;Send character.
          LOAD  s1, 20        ;ASCII " ".
          CALL  xmit          ;Send character.
          LOAD  s1, 52        ;ASCII "R".
          CALL  xmit          ;Send character.
          LOAD  s1, 75        ;ASCII "u".
          CALL  xmit          ;Send character.
          LOAD  s1, 6C        ;ASCII "l".
          CALL  xmit          ;Send character.
          LOAD  s1, 65        ;ASCII "e".
          CALL  xmit          ;Send character.
```

UART Transmit Code, Continued

```
            LOAD  s1, 73        ;ASCII "s".
            CALL  xmit          ;Send character.
            LOAD  s1, 21        ;ASCII "!".
            CALL  xmit          ;Send character.
            LOAD  s1, 21        ;ASCII "!".
            CALL  xmit          ;Send character.
            LOAD  s1, 21        ;ASCII "!".
            CALL  xmit          ;Send character.
            LOAD  s1, 20        ;ASCII " ".
            CALL  xmit          ;Send character.

            JUMP start

; Routine to transmit data via RS-232
; First check UART status.
xmit:       INPUT s0, 00        ;read uart status.
            AND   s0, 80        ;Buffer full mask.
            JUMP  NZ, xmit      ;Poll if buffer is full.

; Buffer is not full, transmit byte in s1
            OUTPUT s1, 00       ;Ship byte.  Don't care about addx.
            RETURN
```