University Bremerhaven

INSTITUTE FOR AUTOMATION AND ELECTR. ENG.

Prof. Dr.-Ing. Kai Mueller

Sample Test System-on-Chip Design ST-1

Time 120 minutes
– class documents allowed –

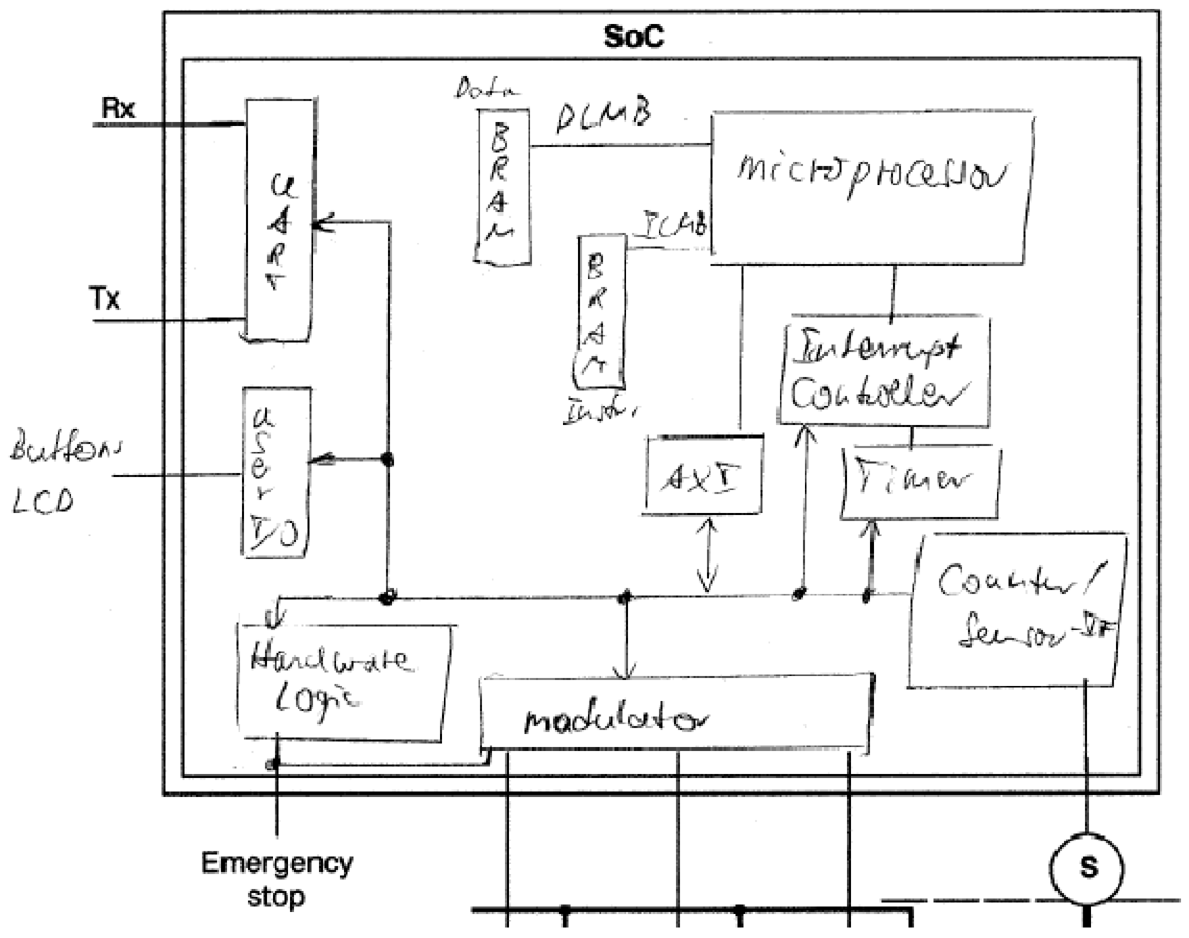
Solutions

(1) General SoC Questions

(1.1) – answers can be found in the class/lab documentation.

(2) SoC Block Diagram for an Industrial Control System

(2.1) Block diagram of SoC components (user logic and processing system):



(1) = 6		(2) = 6		(3) = 6	(4) = 6	(5) = 6			Σ = 30
1.0 ≥ 29	1.3 ≥ 27	1.7 ≥ 26	2.0 ≥ 24	2.3 ≥ 23	2.7 ≥ 21	3.0 ≥ 20	3.3 ≥ 18	3.7 ≥ 17	4.0 ≥ 15

(3) DSP

$F = [\,4.231 \;\; -7.323 \;\; 14.2 \;\; 9.43\,]\,.$

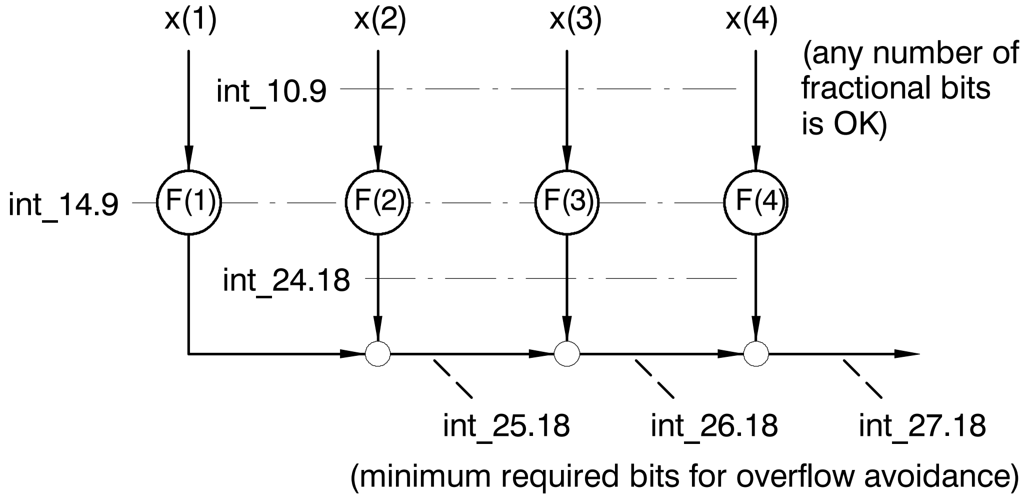
For the coefficients 14 bits are available; data is stored in 10 bit words.

(3.1) Absolute maximum value in F is $|F(3)| = 14.2\,.$ This magnitude requires (as a signed integer ceil(ld(14.2))+1 = 5 bits. Thus we have 9 bits for the fractional bits resulting in int14.9 format.
Range: $-\,2^{14-1}\,2^{-9}\;...\;(2^{14-1}-1)\,2^{-9} \Rightarrow\; -16\;...\;+15.998$

(3.2) Decimal weights

b13	b12		b9	b8		b0
-2^{+4} $=-16$	$+2^{+3}$ $=+8$	• • •	$+2^{+0}$ $=+1$	$+2^{-1}$ $=+½$	• • •	$+2^{-9}$ $=0.002$

(3.3) Block diagram for the product $u = F\,x$:



(4) SoC Hardware

PI Controller with $p\_gain = 0.8$ and $i\_gain = 0.15$ (word size of 16 bits).

(4.1) *p_gain* and *i_gain* in int16.14 format:

$$p\_gain = \text{round}(0.8\ 2^{14}) = 13107$$
$$i\_gain = \text{round}(0.15\ 2^{14}) = 2458$$

(4.2) Analyze the hardware system:

– MicroBlaze CPU (1 CPU),
– the MicroBlaze requires reset controller (`proc_sys_reset_0`) and a clock generator (`clock_generator_0`). Not attached on any bus, directly connected to MicroBlaze,
– four bus systems (AXI-full, AXI-lite, Data Local Memory Bus, Instruction, Local Memory Bus). All busses are driven by MicroBlaze as master,
– RAM controller for Block Memory (`microblaze_0_d_bram_ctl` and `microblaze_0_i_bram_ctl`) and Block memory itself (`microblaze_0_bram_block`),
– DDR2 memory controller for external memory (`MCB_DDR2`) attached to AXI-full,
– Rest of modules are attached to AXI-lite:
  – Debug module (`debug_module`) for downloading and SW debug,
  – Timer (`axi_timer_0`),
  – Interrupt controller for timer interrupts (`microblaze_0_intc`),
  – UART (`RS232_Uart_1`),
  – PI controller in hardware [as user IP] => (`pictrl_0`).

(4.3) `userlogic.vhd` for PI controller hardware (wizard generated code not shown/not required, only ARCHITECTURE section):

```
--USER signal declarations added here, as needed for user
    logic
  CONSTANT  PI_IN_SIZE : integer := 16;
  SUBTYPE  SHORT_INT is integer RANGE -32768 to 32767;
  CONSTANT  P_GAIN : SHORT_INT := 13107;
  CONSTANT  I_GAIN : SHORT_INT :=  2458;
  TYPE state_type is (pi_idle, pi_muladd, pi_integ);
  signal state, next_state : state_type;
  signal Xint : integer := 0;
  signal pi_out : integer;
  --USER logic implementation added here
  pihardw : PROCESS(Bus2IP_Clk, slv_reg_write_sel, Xint,
                    Bus2IP_Resetn, state) IS
  BEGIN
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Resetn = '0' then
               Xint <= 0;
               state <= pi_idle;
          else
               CASE state IS
                 WHEN pi_idle =>
                   IF slv_reg_write_sel = "10" THEN
                      state <= pi_muladd;
                    END IF;
                 WHEN pi_muladd =>
                   pi_out <= P_GAIN *
 conv_integer(signed(slv_reg0(PI_IN_SIZE-1 downto 0))) + Xint;
                     state  <= pi_integ;
                 WHEN pi_integ =>
                   Xint   <= I_GAIN *
 conv_integer(signed(slv_reg0(PI_IN_SIZE-1 downto 0))) + Xint;
                     state  <= pi_idle;
               END CASE;
          end if;
    end if;
  END PROCESS pihardw;
-- implement slave model software accessible register(s)
      read mux
SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0,
slv_reg1, slv_reg2, slv_reg3,pi_out, Xint ) is
  begin
    case slv_reg_read_sel is
      when "10" => slv_ip2bus_data <=
           conv_std_logic_vector(pi_out, 32);
      when "01" => slv_ip2bus_data <=
           conv_std_logic_vector(Xint, 32);
      when others => slv_ip2bus_data <= (others => '0');
    end case;
  end process SLAVE_REG_READ_PROC;
```

(4.4) Explain major elements for PI algorithm from synthesis report:

The algorithm required two multiplications and to adders

$$pi\_out[k] = p\_gain[k]\, u[k] + integ[k],$$
$$integ[k+1] = i\_gain[k]\, u[k] + integ[k],$$

The synthesis reports verifies this. the 16x13-bit multiplier is for the product with `i_gain` (requires only 13 bits as signed number) and the 16x15-bit multiplier is for the product with `p_gain`.

Multiplier and adder are implemented as DSP48 blocks (`DSP48A1s` hardware multiplier and adder) and not in discrete logic using LUTS and FFs (CLBs).

(5) SoC Software

(5.1) Driver for your PI controller hardware from (4):

```
#include <stdio.h>
#include "platform.h"
#include "xuartlite_l.h"
#include "xil_types.h"
#include "xparameters.h"
void print(char *str);

#define  PIREG(k)
          (*(volatile s32 *)(XPAR_PICTRL_0_BASEADDR + 4 * k))
```

```
int main()
{
    unsigned char cmd;

    init_platform();
    print("-- PI (Hardware) Test V0.0a ---\n");
    PIREG(0) = 1;
     do {
          cmd = UARTgetchar();
          xil_printf("[integr] = %d\n", PIREG(1));
          xil_printf("[pi_out] = %d\n", PIREG(0));
          PIREG(0) = 1;
     }
    } while (cmd != 'x');
    PIREG(0) = 0;
    print("Thank you for using PI.\n");
    cleanup_platform();
    return 0;
}
```

The output is the step response of a PI controller with data from (4.1). In every step the integrator is increased by `i_gain`.

(5.2) Analyzing program code (most important sections only):
– program code (.text) located at base address of DDR2 (0xA8000000)
– all data section are also located in DDR2 memory
- .rodata = read-only,
- .bss, .data
- dynamic memory region (.heap)
- stack (.stack)

Since program code is only 0x013b8 in size (appr. 5 kBytes) it could be moved to BRAM which will be much faster than DDR.
Same is true for all data sections up to 8 kBytes for data and program code. In this case no more cache memory is required.
Changing the section for code and data is specified by the link script.