9. Appendix

9.1. VHDL codes

9.1.1. a2tobin

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

entity a2tobin is
    port(
    a2: in std_logic_vector (13 downto 0);
    clk: in std_logic;
    binary: out std_logic_vector (13 downto 0)
    );
end a2tobin;

architecture behaviour of a2tobin is

    signal binary_reg: std_logic_vector (13 downto 0);

    begin

    process (clk)
        begin
        binary_reg <= not(a2+"10000000000000"-"00000000000001");
        --Same as:
        --aux1 <= a2 + "10000000000000";
        --aux2 <= aux1 - "00000000000001";
        --binary_reg <= not aux2;
    end process;

    binary <= binary_reg;

end behaviour;
```

9.1.2. bintoa2

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

entity bintoa2 is
    port(
    binary: in std_logic_vector (13 downto 0);
    clk: in std_logic;
```

```
    a2: out std_logic_vector (13 downto 0)
    );
end bintoa2;

architecture behaviour of bintoa2 is

    signal a2_reg: std_logic_vector (13 downto 0);

    begin

    process (clk)
            begin
        a2_reg <= not(binary(13)) & binary(12 downto 0);
    end process;

    a2 <= a2_reg;

end behaviour;
```

9.1.3. ADAinterfacing

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

entity ADAinterfacing is

    port(
        clkADC : in std_logic;
        clkDAC : in std_logic;
        PLL_ADC : out std_logic_vector (1 downto 0);
        PLL_DAC : out std_logic_vector (1 downto 0);
        DAC_mode : out std_logic;
        DAC_WR_A : out std_logic;
        DAC_OE_A : out std_logic;
        DAC_WR_B : out std_logic;
        DAC_OE_B : out std_logic;
        ADC_OTRA : in std_logic;
        ADC_OTRB : in std_logic;
        OTR_LEDS : out std_logic_vector(1 downto 0);
        ADC_POWERon : out std_logic;
        DAC_dataAin : in std_logic_vector (13 downto 0);
        DAC_dataBin : in std_logic_vector (13 downto 0);
        DAC_dataAout : out std_logic_vector (13 downto 0);
        DAC_dataBout : out std_logic_vector (13 downto 0);
        ADC_dataAin : in std_logic_vector (13 downto 0);
        ADC_dataBin : in std_logic_vector (13 downto 0);
        ADC_dataAout : out std_logic_vector (13 downto 0);
        ADC_dataBout : out std_logic_vector (13 downto 0)
        );

end entity;
```

```
architecture behaviour of ADAinterfacing is

    begin

    PLL_ADC(0) <= clkADC;
    PLL_DAC(0) <= clkDAC;
    PLL_ADC(1) <= clkADC;
    PLL_DAC(1) <= clkDAC;
    DAC_mode <= '1';
    DAC_WR_A <= clkDAC;
    DAC_OE_A <= '0';
    DAC_WR_B <= '0';
    DAC_OE_B <= '0';
    ADC_POWERon <= '1';
    DAC_dataAout <= DAC_dataAin;
    DAC_dataBout <= DAC_dataBin;
    ADC_dataAout <= ADC_dataAin;
    ADC_dataBout <= ADC_dataBin;

end architecture;
```

9.1.4. fsGeneratorAndSelector

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


entity fsGeneratorAndSelector is
    generic(
        cte128 : integer := 12;
        cte192 : integer := 8;
        cte256 : integer := 6;
        cte384 : integer := 4;
        sel128 : std_logic_vector(3 downto 0) := "0000";
        sel192 : std_logic_vector(3 downto 0) := "0001";
        sel256 : std_logic_vector(3 downto 0) := "0010";
        sel384 : std_logic_vector(3 downto 0) := "0011";
        sel1024KHz : std_logic_vector(3 downto 0) := "0100";
        sel4096KHz : std_logic_vector(3 downto 0) := "0101"
        );
    port(
        clkfs : in std_logic;
        clk50 : in std_logic;
        fs_selector : in std_logic_vector(3 downto 0);
        fs1024KHz : in std_logic;
        fs4096KHz : in std_logic;
        --rst_n : in std_logic;
        fs : out std_logic
        );
end entity;

architecture behaviour of fsGeneratorAndSelector is
    signal fs128, fs192, fs256, fs384 : std_logic := '0';
```

```
    signal counter128, counter192, counter256, counter384: integer range 0 to 151 := 0;
        begin
            process(clkfs)
                begin
                if (clkfs'event and clkfs = '1') then

                    if (counter128 < ((cte128/2) - 1)) then
                        counter128 <= counter128 + 1;
                    else
                        fs128 <= not(fs128);
                        counter128 <= 0;
                    end if;

                    if (counter192 < ((cte192/2) - 1)) then
                        counter192 <= counter192 + 1;
                    else
                        fs192 <= not(fs192);
                        counter192 <= 0;
                    end if;

                    if (counter256 < ((cte256/2) - 1)) then
                        counter256 <= counter256 + 1;
                    else
                        fs256 <= not(fs256);
                        counter256 <= 0;
                    end if;

                    if (counter384 < ((cte384/2) - 1)) then
                        counter384 <= counter384 + 1;
                    else
                        fs384 <= not(fs384);
                        counter384 <= 0;
                    end if;

                end if;
            end process;


            fs <= fs128 when fs_selector = sel128 else
                    fs192 when fs_selector = sel192 else
                    fs256 when fs_selector = sel256 else
                    fs384 when fs_selector = sel384 else
                    fs1024KHz when fs_selector = sel1024KHz else
                    fs4096KHz when fs_selector = sel4096KHz else
                    fs4096KHz;



end architecture;
```

9.1.5. comparator

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

entity comparator is
    port(
        clk : in std_logic;
        rst_n : in std_logic;
        dataIn : in std_logic_vector (13 downto 0); --compares up to 4096
        filterLength : in std_logic_vector (13 downto 0);
        comparation : out std_logic
        );
end entity;

architecture behaviour of comparator is

    signal comp : std_logic := '0';

    begin

    process (clk, rst_n)
        begin
        if (rst_n = '0') then
            comp <= '0';
        else
            if (clk'event and clk = '1') then
                if (dataIn = std_logic_vector((unsigned(filterLength)+3-1))) then
                --seria para 2047, pero hay que compensar los
                --tres registros (mult,acum y registro de salida)
                    comp <= '1';
                else
                    comp <= '0';
                end if;
            end if;
        end if;
    end process;
    comparation <= comp;
end architecture;
```

9.1.6. RegComp

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

entity RegComp is
    port(
        clk : in std_logic;
        input : in std_logic;
        output : out std_logic
        );
end entity;
```

```
architecture behaviour of RegComp is
    signal Reg1 : std_logic := '0';
    signal Reg2 : std_logic := '0';

    begin

    process(clk)
    begin
        if (clk'event and clk = '1') then
            Reg1 <= input;
            Reg2 <= Reg1;
        end if;
    end process;
    output <= Reg2;
end architecture;
```

9.1.7. resetControler

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

  entity resetControler is
    port(
        enable : in std_logic;
        resetButton : in std_logic;
        reset_n : out std_logic
        );
end entity;


architecture behaviour of resetControler is

    begin

    reset_n <= '1' when ((resetButton = '1') and (enable = '1'))
                    else '0';

end architecture;
```

9.1.8. RegStability

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

entity RegStability is
    port(
        clk : in std_logic;
        input : in std_logic_vector (41 downto 0);
        output : out std_logic_vector (41 downto 0)
```

```
        );
end entity;


architecture behaviour of RegStability is
    signal Reg1 : std_logic_vector(41 downto 0) := std_logic_vector(to_signed(0,42));
    signal Reg2 : std_logic_vector(41 downto 0) := std_logic_vector(to_signed(0,42));

    begin

    process(clk)
    begin
        if (clk'event and clk = '1') then
            Reg1 <= input;
            Reg2 <= Reg1;
        end if;
    end process;
    output <= Reg2;
end architecture;
```

9.1.9. storageUnit

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

entity storageUnit is
    generic(
        NUM_BITS_DATA : natural := 80
        );

    port(
        clk : in std_logic;
        rst_n : in std_logic;
        load : in std_logic;
        read_request : in std_logic;
        dataQ : in std_logic_vector (41 downto 0);
        dataI : in std_logic_vector (41 downto 0);
        filterLength : in std_logic_vector (13 downto 0);
        empty : out std_logic;
        dataByte : out std_logic_vector (7 downto 0)
        );

end entity;


architecture behaviour of storageUnit is

    type states is (waiting, reading);
    signal state : states := waiting;
    signal counter : natural range 0 to 10 := 0;
    signal byte : std_logic_vector (7 downto 0) := (others => '0');
    signal registeredInput : std_logic_vector (63 downto 0) := (others => '0');
    signal noData : std_logic := '1';
```

```
    signal filterLenReg : std_logic_vector (13 downto 0):=
std_logic_vector(to_unsigned(2048,14));

    begin

    process (clk,rst_n)

        begin
        if (rst_n = '0') then
            state <= waiting;
            noData <= '1';
            counter <= 0;
            byte <= (others => '0');
            registeredInput <= (others => '0');
            filterLenReg <= std_logic_vector(to_unsigned(2048,14));

        elsif (clk'event and clk = '1') then

            filterLenReg <= filterLength;

            case state is

                when waiting =>
                    if (load = '1') then
                        state <= reading;
                        counter <= 0;
                        byte <= (others => '0');
                                noData <= '0';

                        case filterLenReg is
                            when std_logic_vector(to_unsigned(8192,14)) =>
                                registeredInput(63 downto 61) <= (others => dataI(41));
                                registeredInput(60 downto 32) <=  dataI(41 downto 13);
                                registeredInput(31 downto 29) <= (others => dataQ(41));
                                registeredInput(28 downto 0) <=  dataQ(41 downto 13);
                            when std_logic_vector(to_unsigned(4096,14)) =>
                                registeredInput(63 downto 62) <= (others => dataI(41));
                                registeredInput(61 downto 32) <=  dataI(41 downto 12);
                                registeredInput(31 downto 30) <= (others => dataQ(41));
                                registeredInput(29 downto 0) <=  dataQ(41 downto 12);
                            when std_logic_vector(to_unsigned(2048,14)) =>
                                registeredInput(63) <= dataI(41);
                                registeredInput(62 downto 32) <=  dataI(41 downto 11);
                                registeredInput(31) <= dataQ(41);
                                registeredInput(30 downto 0) <=  dataQ(41 downto 11);
                            when std_logic_vector(to_unsigned(1024,14)) =>
                                registeredInput(63 downto 32) <=  dataI(41 downto 10);
                                registeredInput(31 downto 0) <=  dataQ(41 downto 10);
                            when others =>
                                registeredInput(63) <= dataI(41);
                                registeredInput(62 downto 32) <=  dataI(41 downto 11);
                                registeredInput(31) <= dataQ(41);
                                registeredInput(30 downto 0) <=  dataQ(41 downto 11);
                        end case;
```

```
                else
                    state <= state;
                    counter <= 0;
                    byte <= (others => '0');
                    registeredInput <= (others => '0');
                    noData <= '1';
                end if;

            when reading =>
                if read_request = '1' then
                    counter <= counter + 1;
                    byte <= registeredInput((((counter+1)*8)-1) downto counter*8);
                    if counter = 7 then
                        state <= waiting;
                    else
                        state <= reading;
                    end if;
                    registeredInput <= registeredInput;
                    noData <= '0';
                else
                    state <= state;
                    counter <= counter;
                    byte <= (others => '0');
                    registeredInput <= registeredInput;
                    noData <= '0';
                end if;

            when others =>
                state <= waiting;
                counter <= 0;
                byte <= (others => '0');
                registeredInput <= (others => '0');
                noData <= '1';

        end case;

    end if;

end process;

dataByte <= byte;
empty <= noData;
end architecture;
```

9.1.10. uart_manager

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

entity uart_manager is
```

```
    port(
        clk : in std_logic;
        rst_n : in std_logic;
        mem_empty : in std_logic;
        uart_busy : in std_logic;
        rd_mem : out std_logic;
        uart_load : out std_logic
        );

end entity;


architecture behaviour of uart_manager is

    type states is (ready, load, uartPreBusy, uartBusy);
    signal state : states := ready;

    begin

    process (clk,rst_n)

        begin

        if (rst_n = '0') then
            state <= ready;
            rd_mem <= '0';
            uart_load <= '0';

        elsif (clk'event and clk = '1') then

            case state is

                when ready =>
                    if (mem_empty = '0' and uart_busy = '0') then
                        state <= load;
                        rd_mem <= '1';
                        uart_load <= '0';
                    else
                        state <= state;
                        rd_mem <= '0';
                        uart_load <= '0';
                    end if;

                when load =>
                    state <= uartPreBusy;
                    rd_mem <= '0';
                    uart_load <= '1';

                when uartPreBusy =>
                    state <= uartBusy;

                when uartBusy =>
                    rd_mem <= '0';
                    uart_load <= '0';
```

```
                if (uart_busy = '1') then
                    state <= state;
                else
                    state <= ready;
                end if;

            when others =>
                state <= ready;
                rd_mem <= '0';
                uart_load <= '0';

        end case;

    end if;

end process;

end architecture;
```

9.1.11. uart_selector

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

entity uart_selector is


    port(
        clk : in std_logic;
        rst_n : in std_logic;
        enable : in std_logic;
        manager_byte : in std_logic_vector (7 downto 0);
        manager_load : in std_logic;
        nios : in std_logic_vector (8 downto 0);
        to_matlab : out std_logic_vector (8 downto 0)
        );

end entity;


architecture behaviour of uart_selector is
    signal to_matlab_reg : std_logic_vector (8 downto 0) := (others => '0');
    begin


    process(clk,rst_n)
        begin
        if (rst_n = '0') then
            to_matlab_reg <= (others => '0');
        elsif (clk'event and clk = '1') then
            if enable = '0' then
                to_matlab_reg <= nios;
            else
```

```
                to_matlab_reg (7 downto 0) <= manager_byte;
                to_matlab_reg (8) <= manager_load;
            end if;
        end if;
    end process;

    to_matlab <= to_matlab_reg;

end architecture;
```

9.1.12. uart

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

entity uart is

    generic(
    NUM_BITS_TX : natural := 8;
     NUM_BITS_RX : natural := 8;
    num_baud_clock_ticks_bits : natural := 13
    );

    port(
        UART_clk : in std_logic;
        UART_rst_n : in std_logic;
        UART_dataTxIn : in std_logic_vector((NUM_BITS_TX - 1) downto 0);
        UART_dataRxIn : in std_logic;
        UART_txLoad : in std_logic;
        UART_dataTxOut : out std_logic;
        UART_dataRxOut : out std_logic_vector((NUM_BITS_RX - 1) downto 0);
        UART_busy : out std_logic;
        UART_dataAvailable : out std_logic
        );

end entity;


architecture behaviour of uart is


    component uartTx is

        generic(
        NUM_BITS_TX : natural := 8;
        num_baud_clock_ticks_bits : natural := 13
        );

        port(
            UARTclk : in std_logic;
            UARTrst_n : in std_logic;
            UARTload : in std_logic;
            UARTtxData : in std_logic_vector((NUM_BITS_TX - 1) downto 0);
```

```
        UARTbusy : out std_logic;
        UARTtx : out std_logic
        );

end component;


component uartRx is

    generic(
        NUM_BITS_RX : natural := 8;
        num_baud_clock_ticks_bits : natural := 13
        );

    port(
        UARTclk : in std_logic;
        UARTrst_n : in std_logic;
        UARTdataIn : in std_logic;
        UARTdataOut : out std_logic_vector((NUM_BITS_RX - 1) downto 0);
        UARTdataAvailable : out std_logic
        );

end component;



signal met_est_reg1, met_est_reg2 : std_logic := '1';


begin

    instuartTx: uarTtx
        port map(
            UARTclk => UART_clk,
            UARTrst_n => UART_rst_n,
            UARTload => UART_txLoad,
            UARTtxData => UART_dataTxIn,
            UARTbusy => UART_busy,
            UARTtx => UART_dataTxOut
            );

    instuartRx: uartRx
        port map(
            UARTclk => UART_clk,
            UARTrst_n => UART_rst_n,
            UARTdataIn => met_est_reg2,
            UARTdataOut => UART_dataRxOut,
            UARTdataAvailable => UART_dataAvailable
            );

    process (UART_clk, UART_rst_n)
        begin
        if (UART_rst_n = '0') then
            met_est_reg1 <= '1';
```

```
            met_est_reg2 <='1';
        elsif (UART_clk'event and UART_clk ='1') then
            met_est_reg1 <= UART_dataRxIn;
            met_est_reg2 <= met_est_reg1;
        end if;
    end process;


end architecture;
```

9.1.13. smTx

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

entity smTx is

  generic(
    NUM_BITS_TX : natural := 8
    );

    port(
        clk : in std_logic;
        reset : in std_logic;
        load : in std_logic;
        enable : in std_logic;
        txData : in std_logic_vector (7 downto 0);
        baud_rate_generator_e : out std_logic;
        busy : out std_logic;
        tx : out std_logic
        );

end entity;

architecture behaviour of smTx is

    signal counter : unsigned (2 downto 0) := (Others => '0');
    type states is (idle, ready, start_bit, shift, stop_bit, finished);
    signal state : states := idle;
    signal byte : std_logic_vector (7 downto 0);
    signal bittx : std_logic;

    begin

    process (clk)
      begin
      if (reset = '0') then
        byte <= (Others => '0');
        busy <= '0';
        state <= idle;
        baud_rate_generator_e <= '0';
        bittx <= '1';
        elsif (clk'event and clk = '1') then
```

```
case state is
    when idle =>
        if load = '1' then
            baud_rate_generator_e <= '1';
            busy <= '1';
            state <= ready;
            byte <= txData;
        else
            baud_rate_generator_e <= '0';
            busy <= '0';
            state <= idle;
            byte <= (Others => '0');
        end if;
        bittx <= '1';

    when ready =>
        if (enable = '1') then
        state <= start_bit;
        end if;
        bittx <= '1';

    when start_bit =>
        if (enable = '1') then
            state <= shift;
        end if;
        counter <= (others => '0');
        bittx <= '0';

    when shift =>--contadores
        if (enable = '1') then
            if counter = (NUM_BITS_TX -1) then
                state <= stop_bit;
                counter <= (others => '0');
            else
                counter <= counter + 1;
            end if;
            bittx <= byte(0);
            byte(0) <= byte(1);
            byte(1) <= byte(2);
            byte(2) <= byte(3);
            byte(3) <= byte(4);
            byte(4) <= byte(5);
            byte(5) <= byte(6);
            byte(6) <= byte(7);
            byte(7) <= '0';
        end if;
        bittx <= byte(0);

    when stop_bit =>
        if (enable = '1') then
            state <= idle;
            baud_rate_generator_e <= '0';
        end if;
        bittx <= '1';
```

```
                when others =>
                    state <= idle;
                    byte <= (others => '0');
                    bittx <= '1';
                    busy <= '0';
                    baud_rate_generator_e <= '0';

            end case;
        end if;
    end process;

    tx <= bittx;

end architecture;
```

9.1.14. smRx

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

entity smRx is
generic(
   NUM_BITS_RX : natural := 8
   );

    port(
        clk : in std_logic;
        reset : in std_logic;
        rxBit : in std_logic;
        enable : in std_logic;
        baud_rate_gen_e : out std_logic;
        ready : out std_logic;
        rxByte : out std_logic_vector ((NUM_BITS_RX-1) downto 0)
        );
end entity;

architecture behaviour of smRx is

    type states is (idle, filter1, start, capture1, capture2, capture3, capture4, capture5,
capture6, capture7, capture8, stop);
    signal state : states := idle;
    signal byte : std_logic_vector ((num_BITS_RX-1) downto 0);
    signal filterCount : integer range 0 to 100 := 0;

    begin

    process(clk)

    begin

        if (reset = '0') then
            state <= idle;
```

```
            ready <= '0';
            baud_rate_gen_e <= '0';
            byte <= (others => '0');
                filterCount <= 0;
        elsif (clk'event and clk = '1') then
            case state is

                when idle =>
                    if (rxBit = '0') then
                        state <= filter1;
                    else
                        state <=state;
                    end if;
                    byte <= (others => '0');
                    ready <= '0';
                    baud_rate_gen_e <= '0';
                    filterCount <= 0;

                when filter1 =>
                    if (rxBit = '0') then
                        filterCount <= filterCount+1;
                        if (filterCount = 30) then
                            state <= start;
                        else
                            state <= state;
                        end if;
                    else
                        state <=idle;
                    end if;
                    byte <= (others => '0');
                    ready <= '0';
                    baud_rate_gen_e <= '0';

                when start =>
                    if (enable = '1') then
                        state <= capture1;
                    else
                        state <= state;
                    end if;
                    byte <= (others => '0');
                    ready <= '0';
                    baud_rate_gen_e <= '1';

                when capture1 =>
                    if (enable = '1') then
                        byte(0) <= rxBit;
                        --byte((num_BITS_RX-1) downto 1) <= byte((num_BITS_RX-1)
downto 1);
                        state <= capture2;
                    else
                        byte <= byte;
                        state <= state;
                    end if;
                    ready <= '0';
```

```
        baud_rate_gen_e <= '1';


    when capture2 =>
        if (enable = '1') then
            byte(1) <= rxBit;
            state <= capture3;
        else
            byte <= byte;
            state <= state;
        end if;
        ready <= '0';
        baud_rate_gen_e <= '1';

    when capture3 =>
        if (enable = '1') then
            byte(2) <= rxBit;
            state <= capture4;
        else
            byte <= byte;
            state <= state;
        end if;
        ready <= '0';
        baud_rate_gen_e <= '1';

    when capture4 =>
        if (enable = '1') then
            byte(3) <= rxBit;
            state <= capture5;
        else
            byte <= byte;
            state <= state;
        end if;
        ready <= '0';
        baud_rate_gen_e <= '1';

    when capture5 =>
        if (enable = '1') then
            byte(4) <= rxBit;
            state <= capture6;
        else
            byte <= byte;
            state <= state;
        end if;
        ready <= '0';
        baud_rate_gen_e <= '1';

    when capture6 =>
        if (enable = '1') then
            byte(5) <= rxBit;
            state <= capture7;
        else
            byte <= byte;
            state <= state;
```

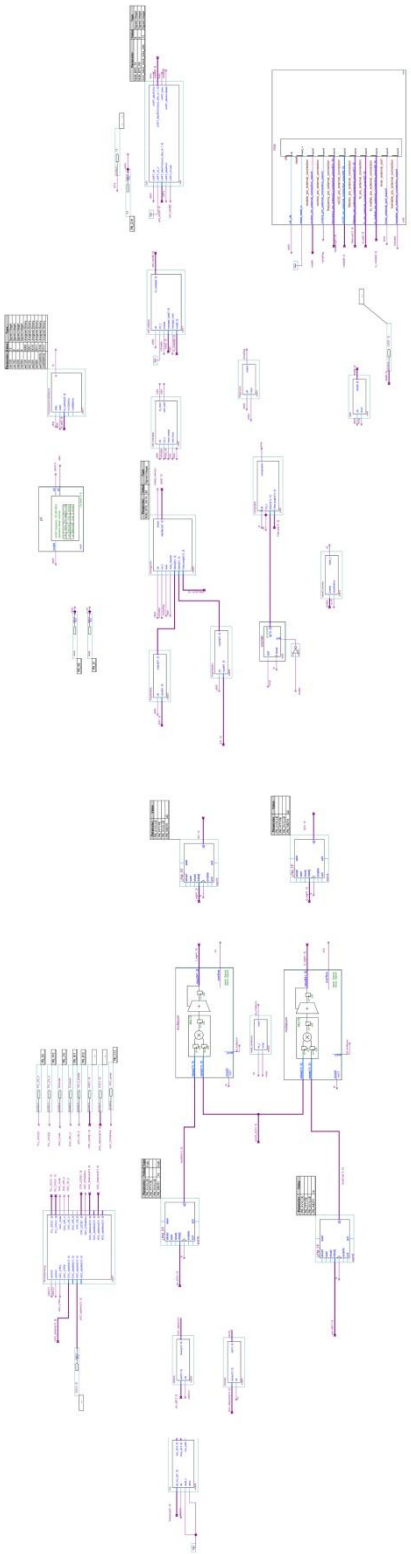
```
                end if;
                ready <= '0';
                baud_rate_gen_e <= '1';

            when capture7 =>
                if (enable = '1') then
                    byte(6) <= rxBit;
                    state <= capture8;
                else
                    byte <= byte;
                    state <= state;
                end if;
                ready <= '0';
                baud_rate_gen_e <= '1';

            when capture8 =>
                if (enable = '1') then
                byte(7) <= rxBit;
                    state <= stop;
                else
                    byte <= byte;
                    state <= state;
                    baud_rate_gen_e <= '1';
                end if;

            when stop =>
                    if (enable = '1') then
                        if (rxBit = '1') then
                            ready <= '1';
                        end if;
                        state <= idle;
                        baud_rate_gen_e <= '0';
                    end if;

            when others =>
                state <= idle;
                byte <= (others => '0');
                ready <= '0';
                baud_rate_gen_e <= '0';

        end case;
    end if;

end process;

rxByte <= byte;

end architecture;
```

9.1.15. baud_rate_generator

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

```
entity baud_rate_generator is

    generic(
        num_baud_clock_ticks_bits : natural := 13
        );

    port(
        clk : in std_logic;
        rst_n : in std_logic;
        enable : in std_logic;
        num_baud_clock_ticks : in std_logic_vector((num_baud_clock_ticks_bits-1)
downto 0);
        trigger_value : in std_logic_vector ((num_baud_clock_ticks_bits-1) downto 0);
        baud_tick: out std_logic
        );

end entity;

architecture behaviour of baud_rate_generator is
    signal tick : std_logic := '0';
    signal clock_counter : unsigned(num_baud_clock_ticks_bits downto 0) := (Others
=>'0');

    begin

    process(clk,rst_n)
        begin
        if (rst_n = '0') then
            tick <= '0';
            clock_counter <= (Others => '0');
        elsif (clk'event and clk = '1') then
            if (enable = '1') then
                if (clock_counter < (unsigned(num_baud_clock_ticks) - 1)) then
                    clock_counter <= clock_counter + 1;
                else
                    clock_counter <= (Others => '0');
                end if;
                if (clock_counter = unsigned(trigger_value)) then
                    tick <= '1';
                else
                    tick <= '0';
                end if;
            else
                clock_counter <= (Others => '0');
                tick <= '0';
            end if;
        end if;
    end process;

    baud_tick <= tick;

end architecture;
```

9.2. FPGA hardware global scheme



9.3. Embedded software

9.3.1. MultifrequencyFRA

```
//Libraries
#include <sys/alt_stdio.h>
#include
<C:\altera\13.1\nios2eds/components/altera_nios2/HAL/inc/io.h>
#include <system.h>
#include <sys/alt_irq.h>
#include <altera_avalon_pio_regs.h>
#include <stdio.h>

//Definitions
#define N_FREQUENCIES 5
#define TIMER_N_cycles 1500000

//Global variables
int n1 = 0;//to count the # of bytes received from MATLAB
int n3 = 0;//to count the # of frequencies measured for each frequency
int waiting = 1;//to synchronize with the timer (when the timer sends
the interruption it's set to 0)
int measuring = 1;//to know when the system has performed one
measurement at one frequency
char byte = 0;//to store the byte received from MATLAB



//Function prototypes
int getInt2 (void);//private function to receive an INT through the
UART (data length 1 Byte)
char pot2 (int x);//private function to check whether X is a power of
2 or not



//Declaration of ISRs as well as their initialization routines

//ISR to receive data from MATLAB
void ISR_matlab_flag(void* context, alt_u32 id)
   {
  byte = IORD_ALTERA_AVALON_PIO_EDGE_CAP(RS232_PIO_BASE);//reads edge
capture register
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(RS232_PIO_BASE, 0);//sets edge
capture register to 0
    n1++;//increments the counter
    }

int init_matlab_flag(void)//its initializing routine
   {
    /* Enable first  interrupt. */
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(RS232_PIO_BASE, 256);//the ninth
bit indicates a new data is coming
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(RS232_PIO_BASE,0);

    /* Register the interrupt handler. */
    return alt_irq_register( RS232_PIO_IRQ, NULL, ISR_matlab_flag );
   }
```

```
//ISR to count the number of samples received of different frequencies
void ISR_counter_flag(void* context, alt_u32 id)
   {
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(CONTROL_PIO_BASE, 0);//we do not
care about the value, it just indicates a sample was transmitted to
MATLAB
     measuring = 0;
     }

 int init_counter_flag(void)//its initializing routine
   {
     /* Enable first  interrupt. */
     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(CONTROL_PIO_BASE, 1);
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(CONTROL_PIO_BASE,0);

     /* Register the interrupt handler. */
     return alt_irq_register( CONTROL_PIO_IRQ, NULL, ISR_counter_flag
);
   }

 //ISR to count the number of samples received for each frequency
 void ISR_timer_flag(void* context, alt_u32 id)
    {
    IOWR_8DIRECT(TIMER_BASE, 0, 0);
    waiting = 0;
    }

 int init_timer_flag(void)
    {
    IOWR_8DIRECT(TIMER_BASE, 4, 3);
    return alt_irq_register( TIMER_IRQ, NULL, ISR_timer_flag );
    }


//Main function
int main()
   {
   int N_SAMPLES = 1001;//to hold the number of samples to obtain
   int n2 = 0;//to know the frequency we are currently measuring
   unsigned int petition = 0;//if there is a request from MATLAB, set
to 1
   unsigned int frequency[5] = {894785,3579139, 5368709, 7158279,
10737418};//frequency of the signal injected for the measurement
   16KHz: 1789570
   unsigned int fsampling[5] = {0, 0, 1, 2, 3};//sampling frequency
   unsigned int filter_length[5] = {1024, 1024, 1024, 1024,
1024};//length of the filter (number of samples for the average)
   char parameters_ok = 1;//Flag: if all the parameters are correct =
1, and an ACK is sent to MATLAB. Otherwise it is set to 0 and a NACK
is sent

     //If the ISRs are correctly registered it goes on
     if (init_timer_flag() != 0 )
        {
        alt_putstr("ISRs Registered unsuccessfully!!!\n");
        return 1;
        }
     else
        {
```

```
        if(init_matlab_flag() != 0 )
           {
           alt_putstr("ISRs Registered unsuccessfully!!!\n");
           return 1;
           }
        else
           {

           if (init_counter_flag() != 0)
              {
              alt_putstr("ISRs Registered unsuccessfully\n");
              return 1;
              }
           else
              {
              alt_putstr("ISRs Registered successfully!!!\n");

              //Initialization of frequency, fs and filter_length values
              IOWR_32DIRECT(FREQUENCY_PIO_BASE, 0, frequency[0]);
              IOWR_16DIRECT(FILTERLEN_PIO_BASE, 0, filter_length[0]);
              IOWR_8DIRECT(FS_PIO_BASE, 0, fsampling[0]);



              while(1)//while the parameters are correct
                 {
                 parameters_ok = 1;//for each iteration we suppose that
parameters are initially correct

                 petition = getInt2();//wait and check for a request

                 if (petition == 1)//if the code received is "request" we
send an ACK, otherwise a NACK
                    {
                    IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 257);//After
writing the value, we should set the ninth bit to 0 again. If not, the
UART will not stop sending values, since this bit indicates there is a
byte to send
                    IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);
                    alt_putstr("Request from user\n");
                    }
                 else
                    {
                    IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 256);
                    IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);
                    alt_putstr("Error\n");
                    }

                 //Acquisition of parameters

                 //Number of samples
                 N_SAMPLES = getInt2();

                 //Limits of # of samples
                 if ((N_SAMPLES > 100000) || (N_SAMPLES < 5))
                    {
                    parameters_ok = 0;
                    }
```

```
            if (parameters_ok == 0)//if parameters are incorrect, we
send a NACK and come back to the while
                {
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 256);
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);
                }
            else
                {//if parameters are correct we send an ACK
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 257);
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);

                n2 = 0;
                n3 = 0;
                measuring = 1;

                N_SAMPLES = N_SAMPLES+1;//The number of ticks received
should be one more of the specified, to give the last sample time
enough to be sent
                //The acquisition system is enabled
                IOWR_8DIRECT(TXENABLE_PIO_BASE, 0, 1);
                //Timer starts
                IOWR_8DIRECT(TIMER_BASE, 4, 7);

                while (n3 < N_SAMPLES)
                  {

                  while(n2 < N_FREQUENCIES)
                    {
                    IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 0);
                    IOWR_32DIRECT(FREQUENCY_PIO_BASE, 0,
frequency[n2]);
                    IOWR_16DIRECT(FILTERLEN_PIO_BASE, 0,
filter_length[n2]);
                    IOWR_8DIRECT(FS_PIO_BASE, 0, fsampling[n2]);
                    IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 1);
                    while(measuring)
                      {
                      IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 1);
                      }
                    measuring = 1;
                    n2++;
                    }
                  IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 0);
                  n2 = 0;
                  while(waiting)
                    {
                    IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 0);//This
instruction does not do anything new, but inserting an instruction
here was needed for the program to work properly
                    }
                  waiting = 1;
                  n3++;
                  }
                IOWR_8DIRECT(TXENABLE_PIO_BASE, 0, 0);
                //Stop and reset the timer
                IOWR_8DIRECT(TIMER_BASE, 4, 11);
                IOWR_32DIRECT(TIMER_BASE, 8, TIMER_N_cycles);
                }
              }
            }
          return 0;
```

```
            }
         }
   }


int getInt2 (void)
   {

   unsigned int number = 0;
   unsigned int i = 0;
   unsigned int x = 0;
   n1 = 0;
      while (i < 4)
         {
         while (n1 < 1)
            {
            IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 0);
            }
         n1 = 0;
         x = ((unsigned int)byte) & 255;//the use of the mask is
completely needed. If the MSB of byte is 1,
         //the cast fill the most significant bits of X with 1's,
despite of being unsigned
         x = x << 8*i;
         number = number + x;
         i++;
         }
   n1 = 0;
   return number;
   }

char pot2 (int x)
   {
   char potencia = 0;
   char i = 0;
      while((i < 32) && (potencia < 2))
      {
      potencia = potencia + (x & 1);
      x = x >> 1;
      i++;
      }
   return potencia;
   }
```

9.3.2. MonofrequencyFRA

```
//Libraries
#include <sys/alt_stdio.h>
#include
<C:\altera\13.1\nios2eds/components/altera_nios2/HAL/inc/io.h>
#include <system.h>
#include <sys/alt_irq.h>
#include <altera_avalon_pio_regs.h>
#include <stdio.h>


//Global variables
int n1 = 0;//to count the # of bytes received from MATLAB
int n2 = 0;//to count the # of samples received from the measurement
system
char byte = 0;//to store the byte received from MATLAB



//Function prototypes
int getInt2 (void);//private function to receive an INT through the
UART (data length 1 Byte)
char pot2 (int x);//private function to check whether X is a power of
2 or not



//Declaration of ISRs as well as their initialization routines

//ISR to receive data from MATLAB
void ISR_matlab_flag(void* context, alt_u32 id)
   {
   byte = IORD_ALTERA_AVALON_PIO_EDGE_CAP(RS232_PIO_BASE);//reads edge
capture register
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(RS232_PIO_BASE, 0);//sets edge
capture register to 0
    n1++;//increments the counter
    }

int init_matlab_flag(void)//its initializing routine
   {
    /* Enable first  interrupt. */
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(RS232_PIO_BASE, 256);//the ninth
bit indicates a new data is coming
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(RS232_PIO_BASE,0);

    /* Register the interrupt handler. */
    return alt_irq_register( RS232_PIO_IRQ, NULL, ISR_matlab_flag );
   }



//ISR to count the number of samples received
void ISR_counter_flag(void* context, alt_u32 id)
   {
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(CONTROL_PIO_BASE, 0);//we do not
care about the value, it just indicates a sample was transmitted to
MATLAB
    n2++;
    }
```

```
 int init_counter_flag(void)//its initializing routine
   {
     /* Enable first  interrupt. */
     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(CONTROL_PIO_BASE, 1);
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(CONTROL_PIO_BASE,0);

     /* Register the interrupt handler. */
     return alt_irq_register( CONTROL_PIO_IRQ, NULL, ISR_counter_flag
);
   }

//Main function
int main()
   {
   int N_SAMPLES = 1001;//to hold the number of samples to obtain
   unsigned int petition = 0;//if there is a request from MATLAB, set
to 1
   unsigned int frequency = 107374182;//frequency of the signal
injected for the measurement
   unsigned int fsampling = 5;//sampling frequency
   unsigned int filter_length = 2048;//length of the filter (number of
samples for the average)
   char parameters_ok = 1;//Flag: if all the parameters are correct =
1, and an ACK is sended to MATLAB. Otherwise it is set to 0 and a NACK
is sended
   char temp;//aux variable

        if(init_matlab_flag() != 0 )
        //If the ISRs are correctly registered it goes on
          {
          alt_putstr("ISRs Registered unsuccessfully!!!\n");
          return 1;
          }
        else
          {

          if (init_counter_flag() != 0)
            {
            alt_putstr("ISRs Registered unsuccessfully\n");
            return 1;
            }
          else
            {
            alt_putstr("ISRs Registered successfully!!!\n");

            //Initialization of frequency and filter_length values
            IOWR_32DIRECT(FREQUENCY_PIO_BASE, 0, frequency);
            IOWR_16DIRECT(FILTERLEN_PIO_BASE, 0, filter_length);


            while(parameters_ok == 1)//while the parameters are correct
              {
              parameters_ok = 1;//for each iteration we suppose that
parameters are initially correct

              petition = getInt2();//wait and check for a request

              if (petition == 1)//if the code received is "request" we
send an ACK, otherwise a NACK
                {
```

```
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 257);//After
writing the value, we should set the ninth bit to 0 again. If not, the
UART will not stop sending values, since this bit
                //indicates there is a byte to send
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);
                alt_putstr("Request from user\n");
                }
             else
                {
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 256);
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);
                alt_putstr("Error\n");
                }

             //Acquisition of parameters

             //Frequency
             frequency = getInt2();
             /*Limits of frequency
             if ((frequency > 1000000) || (frequency < 1000))
                {
                parameters_ok = 0;
                }*/

             //Sampling frequency
             fsampling = getInt2();
             /*Limits of sampling frequency
             switch ( fsampling )
                {
                case 10000:
                   IOWR_8DIRECT(FS_PIO_BASE, 0, 1);
                   break;
                case 500:
                   IOWR_8DIRECT(FS_PIO_BASE, 0, 2);
                   break;
                default:
                   IOWR_8DIRECT(FS_PIO_BASE, 0, 3);
                   parameters_ok = 0;
                   break;
                }*/

             //Filter_length
             filter_length = getInt2();


             //Number of samples
             N_SAMPLES = getInt2();

             //Check if # of samples is power of 2
             temp = pot2(filter_length);
                if (temp != 1)
                   {
                   parameters_ok = 0;
                   }

             //Limits of # of samples
             if ((N_SAMPLES > 100000) || (N_SAMPLES < 5))
                {
                parameters_ok = 0;
                }
```

```
            if (parameters_ok == 0)//if parameters are incorrect, we
send a NACK and come back to the while
                {
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 256);
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);
                }
              else
                {//if parameters are correct we send an ACK and pass
the parameters to the system of
                //measurement
                IOWR_16DIRECT(FILTERLEN_PIO_BASE, 0, filter_length);
                IOWR_32DIRECT(FREQUENCY_PIO_BASE, 0, frequency);
                IOWR_8DIRECT(FS_PIO_BASE, 0, fsampling);
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 257);
                IOWR_16DIRECT(TO_MATLAB_PIO_BASE, 0, 0);

                n2 = 0;

                N_SAMPLES = N_SAMPLES+1;//The number of ticks received
should be one more of the specified, to give the last sample time
enough to be sended
                //The acquisition system is enabled
                IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 1);
                while (n2 < N_SAMPLES)
                  {
                  IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 1);//This
instruction does not do anything new, but inserting an instruction
here was needed for the program to work properly
                  }
                //Once all the samples have been sended, the
acquisition system is disabled and a new iteration begins
                IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 0);
                }
              }
            }
          return 0;
          }
  }


int getInt2 (void)
  {

  unsigned int number = 0;
  unsigned int i = 0;
  unsigned int x = 0;
  n1 = 0;
    while (i < 4)
      {
      while (n1 < 1)
        {
        IOWR_8DIRECT(ENABLE_PIO_BASE, 0, 0);
        }
      n1 = 0;
      x = ((unsigned int)byte) & 255;//the use of the mask is
completely needed. If the MSB of byte is 1, the cast fill the most
significant bits of X with 1's, despite of being unsigned
      x = x << 8*i;
```

```
        number = number + x;
        i++;
        }
   n1 = 0;
   return number;
   }

char pot2 (int x)
   {
   char potencia = 0;
   char i = 0;
      while((i < 32) && (potencia < 2))
      {
      potencia = potencia + (x & 1);
      x = x >> 1;
      i++;
      }
   return potencia;
   }
```

9.4. Matlab scripts

The most important Matlab scripts used in this project are given in this section.

9.4.1. Script to obtain measurements at the five frequencies described in section 1.2

```
%Debido al hardware diseñado en primer lugar se reciben los 4 bytes
%correspondientes a Q y despues los 4 bytes correspondientes a I

clear all;
close all;
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%                              PARÁMETROS
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
Time=30;%En seg. Si tiempo es = 0, se toma el valor de número de
muestras
Nmuestras=500;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

if Time ~= 0
    Nmuestras=ceil(Time/0.03);
end;

PS=serial('COM5');%En el portátil utilizo este puerto
set(PS,'Baudrate',256000); % se configura la velocidad a 115200
Baudios, seguramente funcione a mas velocidad
set(PS,'StopBits',1); % se configura bit de parada a uno
set(PS,'DataBits',8); % se configura que el dato es de 8 bits, debe
estar entre 5 y 8
set(PS,'Parity','none'); % se configura sin paridad
set(PS,'OutputBufferSize',8); %dd "n" es el número de bytes a enviar
set(PS,'InputBufferSize' ,Nmuestras*2*4*5+1); % "n" es el número de
bytes a recibir
set(PS,'Timeout',60); % 5 segundos de tiempo de espera

fopen(PS);

Q8=[];
I8=[];
Q32=[];
I32=[];
Q48=[];
I48=[];
Q64=[];
I64=[];
Q96=[];
I96=[];

%Request of measurement
fwrite(PS,1,'uint32');
```

```
%Confirmed
msg = fread(PS,1,'uint8');
if msg == 1
    disp('Established');
else
    error('Error');
end;

%Send parameters

fwrite(PS,Nmuestras,'uint32');

%Validation of parameters
msg = fread(PS,1,'uint8');
if msg == 0
    error('Some parameter is wrong');
elseif msg == 1
    disp('ACK');
    else disp('Unusual error');
end;


for i=1:(Nmuestras)

Qi=fread(PS,1,'int');
Q8=[Q8 Qi];
Ii=fread(PS,1,'int');
I8=[I8 Ii];

Qi=fread(PS,1,'int');
Q32=[Q32 Qi];
Ii=fread(PS,1,'int');
I32=[I32 Ii];

Qi=fread(PS,1,'int');
Q48=[Q48 Qi];
Ii=fread(PS,1,'int');
I48=[I48 Ii];

Qi=fread(PS,1,'int');
Q64=[Q64 Qi];
Ii=fread(PS,1,'int');
I64=[I64 Ii];

Qi=fread(PS,1,'int');
Q96=[Q96 Qi];
Ii=fread(PS,1,'int');
I96=[I96 Ii];

i
end;

fclose(PS);
delete(PS);
clear PS;
INSTRFIND
```

```
%Cálculo de parámetros y representación a partir de las señales de
fase y cuadratura


%Cálculo de magnitudes y fases para cada muestra de cada frecuencia
Zmag8_noCal=sqrt(Q8.^2+I8.^2);
Zpha8_noCal=atan(Q8./I8);

Zmag32_noCal=sqrt(Q32.^2+I32.^2);
Zpha32_noCal=atan(Q32./I32);

Zmag48_noCal=sqrt(Q48.^2+I48.^2);
Zpha48_noCal=atan(Q48./I48);

Zmag64_noCal=sqrt(Q64.^2+I64.^2);
Zpha64_noCal=atan(Q64./I64);

Zmag96_noCal=sqrt(Q96.^2+I96.^2);
Zpha96_noCal=atan(Q96./I96);

%Base de tiempos (en ms)

tslot=0.03;
tbase=tslot*linspace(0,Nmuestras-1,Nmuestras);

%Ploteo
figure(1)
subplot(2,1,1)
plot(tbase,Q8)
subplot(2,1,2)
plot(tbase,I8)

figure(2)
subplot(2,1,1)
plot(tbase,Q32)
subplot(2,1,2)
plot(tbase,I32)

figure(3)
subplot(2,1,1)
plot(tbase,Q48)
subplot(2,1,2)
plot(tbase,I48)

figure(4)
subplot(2,1,1)
plot(tbase,Q64)
subplot(2,1,2)
plot(tbase,I64)

figure(5)
subplot(2,1,1)
plot(tbase,Q96)
subplot(2,1,2)
plot(tbase,I96)


%Cálculo de las SNR a 96 kHz
```

```
Si96=(sum(I96cal)/Nmuestras)^2;
Ni96=var(I96cal);
SNRi96=10*log10(Si96/Ni96);

Sq96=(sum(Q96cal)/Nmuestras)^2;
Nq96=var(Q96cal);
SNRq96=10*log10(Sq96/Nq96);

%Almacenamos los valores medios a cada frecuencia en un fichero
Magnitudes=[mean(Zmag8_noCal) mean(Zmag32_noCal) mean(Zmag48_noCal)
mean(Zmag64_noCal) mean(Zmag96_noCal)];
Fases=[mean(Zpha8_noCal) mean(Zpha32_noCal) mean(Zpha48_noCal)
mean(Zpha64_noCal) mean(Zpha96_noCal)];
R=150;
name = strcat('ramonrespiracionTorsoCorazon2',num2str(R),'.txt');
fileID = fopen(name,'w');
fprintf(fileID,'%.8f ',Magnitudes);
fprintf(fileID,'\r\n');
fprintf(fileID,'%.8f ',Fases);
fclose(fileID);
```

9.4.2. Script to obtain weights for calibration

```
function calibration (QnoCal, InoCal, Zref, freq)
%Zref=[R,I]
%Freq en KHz
I = importdata(QnoCal);%imaginaria
R = importdata(InoCal);%real

Mag = sqrt(R.^2+I.^2);
Ph = atan(I./R);

MagAvg = mean(Mag);
PhAvg = mean(Ph);

MagRef = sqrt(Zref(1)^2+Zref(2)^2);
PhRef = atan(Zref(2)/Zref(1));

MagW = MagRef/MagAvg;
PhW = PhRef-PhAvg;

name = strcat('weights',num2str(freq),'.txt');

fileID = fopen(name,'w');
fprintf(fileID,'%.8f\r\n',MagW);
fprintf(fileID,'%.8f',PhW);
fclose(fileID);
```

9.4.3. Script for circle fitting

```
function    [xc,yc,R,a] = circfit(x,y)

   x=x(:);
```

```
y=y(:);
a=[x y ones(size(x))]\[-(x.^2+y.^2)];
xc = -.5*a(1);
yc = -.5*a(2);
R  =  sqrt((a(1)^2+a(2)^2)/4-a(3));
```

9.4.4. Script for plotting figure 5.8

```
%Para graficar, cargar previamente datos en carpeta '150_5segs'
%(load 'a.mat')
close all;

Zmag8_noCal=sqrt(Q8.^2+I8.^2);
Zpha8_noCal=atan(Q8./I8);

Zmag32_noCal=sqrt(Q32.^2+I32.^2);
Zpha32_noCal=atan(Q32./I32);

Zmag48_noCal=sqrt(Q48.^2+I48.^2);
Zpha48_noCal=atan(Q48./I48);

Zmag64_noCal=sqrt(Q64.^2+I64.^2);
Zpha64_noCal=atan(Q64./I64);

Zmag96_noCal=sqrt(Q96.^2+I96.^2);
Zpha96_noCal=atan(Q96./I96);

Nmuestras=167;

hist(Zmag48_noCal,15)
sigma=sqrt(var(Zmag48_noCal));
x=linspace(5.44e6,5.56e6,1000);
y = 30*gaussmf(x,[sigma mean(Zmag48_noCal)]);
hold on;
plot(x,y,'r')
title('Histogram at 48KHz (5secs)')
xlabel('Magnitude(Counts)')
2*sigma*148.3/mean(Zmag48_noCal)%0.9372 ohmios con un 95%, a partirde
1000 muestras
```

9.4.5.Script for plotting figures 5.3-5.7

```
close all;

fileID = fopen('R10.txt','r');
R10 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R15.txt','r');
R15 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R22.txt','r');
R22 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R33.txt','r');
R33 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R47.txt','r');
```

```
R47 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R68.txt','r');
R68 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R100.txt','r');
R100 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R150.txt','r');
R150 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R220.txt','r');
R220 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R330.txt','r');
R330 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R470.txt','r');
R470 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R680.txt','r');
R680 = fscanf(fileID,'%f')
fclose(fileID);
fileID = fopen('R1000.txt','r');
R1000 = fscanf(fileID,'%f')
fclose(fileID);


Mag8KHz=[R10(1) R15(1) R22(1) R33(1) R47(1) R68(1) R100(1) R150(1)
R220(1) R330(1) R470(1) R680(1) R1000(1)];
Pha8KHz=(180/pi)*[R10(6) R15(6) R22(6) R33(6) R47(6) R68(6) R100(6)
R150(6) R220(6) R330(6) R470(6) R680(6) R1000(6)];
Pha8KHz= Pha8KHz+(-1*sign(Pha8KHz)+1)*180;
Mag32KHz=[R10(2) R15(2) R22(2) R33(2) R47(2) R68(2) R100(2) R150(2)
R220(2) R330(2) R470(2) R680(2) R1000(2)];
Pha32KHz=(180/pi)*[R10(7) R15(7) R22(7) R33(7) R47(7) R68(7) R100(7)
R150(7) R220(7) R330(7) R470(7) R680(7) R1000(7)];
Pha32KHz= Pha32KHz+(-1*sign(Pha32KHz)+1)*180;
Mag48KHz=[R10(3) R15(3) R22(3) R33(3) R47(3) R68(3) R100(3) R150(3)
R220(3) R330(3) R470(3) R680(3) R1000(3)];
Pha48KHz=(180/pi)*[R10(8) R15(8) R22(8) R33(8) R47(8) R68(8) R100(8)
R150(8) R220(8) R330(8) R470(8) R680(8) R1000(8)];
Pha48KHz= Pha48KHz+(-1*sign(Pha48KHz)+1)*180;
Mag64KHz=[R10(4) R15(4) R22(4) R33(4) R47(4) R68(4) R100(4) R150(4)
R220(4) R330(4) R470(4) R680(4) R1000(4)];
Pha64KHz=(180/pi)*[R10(9) R15(9) R22(9) R33(9) R47(9) R68(9) R100(9)
R150(9) R220(9) R330(9) R470(9) R680(9) R1000(9)];
Pha64KHz= Pha64KHz+(-1*sign(Pha64KHz)+1)*180;
Mag96KHz=[R10(5) R15(5) R22(5) R33(5) R47(5) R68(5) R100(5) R150(5)
R220(5) R330(5) R470(5) R680(5) R1000(5)];
Pha96KHz=(180/pi)*[R10(10) R15(10) R22(10) R33(10) R47(10) R68(10)
R100(10) R150(10) R220(10) R330(10) R470(10) R680(10) R1000(10)];
Pha96KHz= Pha96KHz+(-1*sign(Pha96KHz)+1)*180;


base=[10.2 15.5 22.3 33.4 47 67.6 98.8 148.3 218.7 322.5 465.1 670.2
987];

figure(1)
subplot(2,1,1)
```

```
plot(base,Mag8KHz,'Marker','o')
title('Measurements at 8KHz')
xlabel('Resistance(Ohms)')
ylabel('Magnitude')
subplot(2,1,2)
plot(base,Pha8KHz,'Marker','o')
xlabel('Resistance(Ohms)')
ylabel('Phase (deg)')
axis([0 1000 0 360])

figure(2)
subplot(2,1,1)
plot(base,Mag32KHz,'Marker','o')
title('Measurements at 32KHz')
xlabel('Resistance(Ohms)')
ylabel('Magnitude')
subplot(2,1,2)
plot(base,Pha32KHz,'Marker','o')
xlabel('Resistance(Ohms)')
ylabel('Phase (deg)')
axis([0 1000 0 360])

figure(3)
subplot(2,1,1)
plot(base,Mag48KHz,'Marker','o')
title('Measurements at 48KHz')
xlabel('Resistance(Ohms)')
ylabel('Magnitude')
subplot(2,1,2)
plot(base,Pha48KHz,'Marker','o')
xlabel('Resistance(Ohms)')
ylabel('Phase (deg)')
axis([0 1000 0 360])

figure(4)
subplot(2,1,1)
plot(base,Mag64KHz,'Marker','o')
title('Measurements at 64KHz')
xlabel('Resistance(Ohms)')
ylabel('Magnitude')
subplot(2,1,2)
plot(base,Pha64KHz,'Marker','o')
xlabel('Resistance(Ohms)')
ylabel('Phase (deg)')
axis([0 1000 0 360])

figure(5)
subplot(2,1,1)
plot(base,Mag96KHz,'Marker','o')
title('Measurements at 96KHz')
xlabel('Resistance(Ohms)')
ylabel('Magnitude')
subplot(2,1,2)
plot(base,Pha96KHz,'Marker','o')
xlabel('Resistance(Ohms)')
ylabel('Phase (deg)')
axis([0 1000 0 360])
```

9.4.6. Script for plotting figures 5.14-5.16

```
close all;

figure(1)
R1=51;
R2=148.6;
f=linspace(1,100,100)*1e3;
w=2*pi*f;
C=56e-9;
Zmag=sqrt((R1+R2+w.^2*C^2*R2^2*R1).^2+(w*C*R2^2).^2)*1./(1+w.^2*C^2*R2
^2);
semilogx(f,Zmag,'g')

f=[8e3 32e3 48e3 64e3 96e3];
mag=[2.338604829442990e+05*148.3/192404.50918908
7.389931607183700e+06*148.3/9069969.24215753
3.367369502277007e+06*148.3/5474982.50271059
2.513054666433730e+06*148.3/4824709.89097784
1.950447531651993e+06*148.3/4490127.80937981];
hold on;
semilogx(f,mag,'o')
title('RC net')
xlabel('Frequency(Hz)')
ylabel('Magnitude(Ohms)')
legend('Theoretical values','Measured values')

Zpha=phase((R1+1i*w*C*R1*R2+R2)./(1+1i*w*C*R2))*180/pi;%Ojo: phase
puede dar errores de pi

figure(2)
f=linspace(1,100,100)*1e3;
semilogx(f,Zpha,'g')
title('RC net')
xlabel('Frequency(Hz)')
ylabel('Phase(deg)')
hold on;
pha=[0.97629839-1.24444391 1.54776672-(-0.96342329)-pi 1.22345424-(-
1.29037245)-pi 1.15208943-(-1.40729206)-pi 1.10714991-(-1.55164370)-
pi]*180/pi;
f=[8e3 32e3 48e3 64e3 96e3];
semilogx(f,pha,'o')
xlabel('Frequency(Hz)')
ylabel('Phase(deg)')
legend('Theoretical values','Measured values')

f=linspace(1,1000000,1000000);
w=2*pi*f;
Zreal=real((R1+1i*w*C*R1*R2+R2)./(1+1i*w*C*R2));
Zimag=imag((R1+1i*w*C*R1*R2+R2)./(1+1i*w*C*R2));
figure(3)
plot(Zreal,Zimag,'g')

Real8KHz=mag(1)*cos(pha(1)*pi/180);
Imag8KHz=mag(1)*sin(pha(1)*pi/180);
Real32KHz=mag(2)*cos(pha(2)*pi/180);
Imag32KHz=mag(2)*sin(pha(2)*pi/180);
Real48KHz=mag(3)*cos(pha(3)*pi/180);
Imag48KHz=mag(3)*sin(pha(3)*pi/180);
```

```
Real64KHz=mag(4)*cos(pha(4)*pi/180);
Imag64KHz=mag(4)*sin(pha(4)*pi/180);
Real96KHz=mag(5)*cos(pha(5)*pi/180);
Imag96KHz=mag(5)*sin(pha(5)*pi/180);
Reales=[Real8KHz Real32KHz Real48KHz Real64KHz Real96KHz];
Reactancias=[Imag8KHz Imag32KHz Imag48KHz Imag64KHz Imag96KHz];
hold on;
plot(Reales,Reactancias,'o')
hold on;
plot(Zreal(19126),Zimag(19126),'Marker','x','Color','r')

[xfit,yfit,Rfit] = circfit(Reales,Reactancias);
hold on
rectangle('position',[xfit-Rfit,yfit-Rfit,Rfit*2,Rfit*2],...
    'curvature',[1,1],'linestyle','-','edgecolor','m');
title(sprintf('Circle fitting: R = %0.1f; Ctr = (%0.1f,%0.1f)',...
    Rfit,xfit,yfit));

hold on;
plot(Zreal(19126),Zimag(19126),'m')

xlabel('Resistance(Ohms)')
ylabel('Reactance(Ohms)')
axis([0 250 -250 0]);
legend('Theoretical values','Measured values','Theoretical Centre
frequency','Circle fitting')
```

9.4.7. Script for plotting figure 5.24

```
load 'matlab.mat'

close all;

Zmag8_noCal=sqrt(Q8.^2+I8.^2);
Zpha8_noCal=atan(Q8./I8);

Zmag32_noCal=sqrt(Q32.^2+I32.^2);
Zpha32_noCal=atan(Q32./I32);

Zmag48_noCal=sqrt(Q48.^2+I48.^2);
Zpha48_noCal=atan(Q48./I48);

Zmag64_noCal=sqrt(Q64.^2+I64.^2);
Zpha64_noCal=atan(Q64./I64);

Zmag96_noCal=sqrt(Q96.^2+I96.^2);
Zpha96_noCal=atan(Q96./I96);

Nmuestras=1000;
tslot=0.03;
tbase=tslot*linspace(0,Nmuestras-1,Nmuestras);

N=10;
b=(1/N)*ones(1,N);
a=1;

figure(1)
```

```
subplot(2,1,1)
plot(tbase,Zmag8_noCal)
y=filtfilt(b,a,Zmag8_noCal);
hold on;
plot(tbase,y,'r')
title('Measurement at 8kHz')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')
legend('Measured signal','Filtered signal')
subplot(2,1,2)
y=detrend(y);
plot(tbase,y,'r')
title('Smoothed signal')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')

figure(2)
subplot(2,1,1)
plot(tbase,Zmag32_noCal)
y=filtfilt(b,a,Zmag32_noCal);
hold on;
plot(tbase,y,'r')
title('Measurement at 32kHz')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')
legend('Measured signal','Filtered signal')
subplot(2,1,2)
y=detrend(y);
plot(tbase,y,'r')
title('Smoothed signal')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')

figure(3)
subplot(2,1,1)
plot(tbase,Zmag48_noCal)
y=filtfilt(b,a,Zmag48_noCal);
hold on;
plot(tbase,y,'r')
title('Measurement at 48kHz')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')
legend('Measured signal','Filtered signal')
subplot(2,1,2)
y=detrend(y);
plot(tbase,y,'r')
title('Smoothed signal')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')


figure(4)
subplot(2,1,1)
plot(tbase,Zmag64_noCal)
y=filtfilt(b,a,Zmag64_noCal);
hold on;
plot(tbase,y,'r')
title('Measurement at 64kHz')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')
```

```
legend('Measured signal','Filtered signal')
subplot(2,1,2)
y=detrend(y);
plot(tbase,y,'r')
title('Smoothed signal')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')

figure(5)
subplot(2,1,1)
plot(tbase,Zmag96_noCal)
y=filtfilt(b,a,Zmag96_noCal);
hold on;
plot(tbase,y,'r')
title('Measurement at 96kHz')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')
legend('Measured signal','Filtered signal')
subplot(2,1,2)
y=detrend(y);
plot(tbase,y,'r')
title('Smoothed signal')
xlabel('Time(secs)')
ylabel('Magnitude(counts)')
```