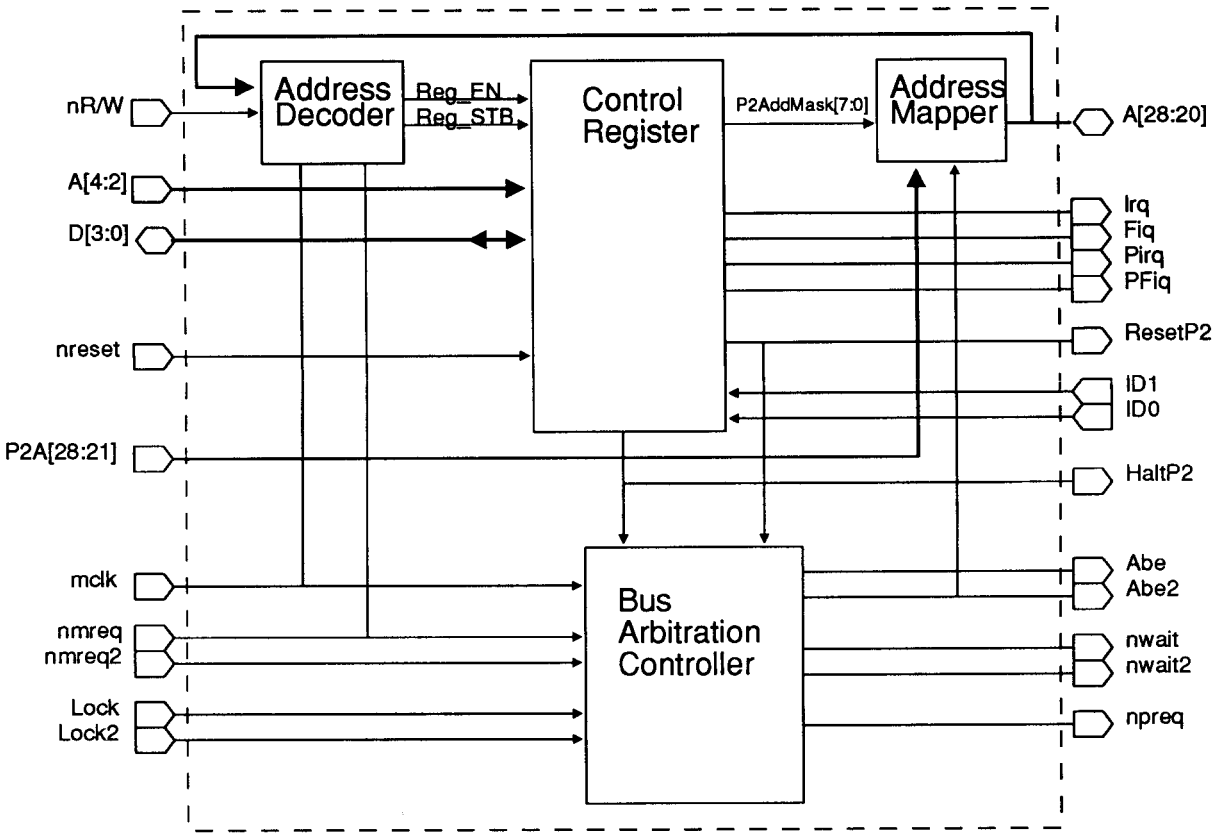
Duet - Medusa 2nd ARM Processor
Functional Specification

Distribution:	COMPANY CONFIDENTIAL
Title:	Duet - Medusa 2nd ARM Processor Functional Specification
Drawing Number:	xxxxxxxx
Issue:	C****LIVE****
Author(s):	Charles Sturman
Date:	15th June 1994
Change Number:	
Last Issue:	B - 4th March 1994

Contents:

1. Functional Description

'Duet' Dual ARM Processor System



The address decoder decodes address bus signals A[28:20] and is activated when an access occurs to memory region 370 0000h to 37F FFFFh. A strobe signal is generated for writing data into the register block and an output enable signal for controlling the tri-stating of data from the registers onto the data bus.

The control register contains two 4 bit address mask registers, four 1 bit interrupt flag registers and a 4 bit control & 1D register. This block is described in the next section.

The address mapper ORs the address mask register outputs with the respective 2nd processor address bits and, under control of abe2, tristates the resulting masked address onto the address bus. In this way, a mask value may be written to the regsietrs and the 2nd processor will commence execution from an abritary address on a 2MB boundary between 1000 0000h and 1FFF FFFh.

Control of the two processors' bus and clock control signals is handled by the bus arbitration state machine. This controls the abe and nwait signals to each processor and generates npreq signals to 1OMD to indicate that the 2nd processor wishes to use the bus. This state machine is described further below.

2. Register Map

Duet Reg register map
Sel<2..0> accesses the control registers (0 to 6) as follows:

Sel	Register set on Write	Default
0	Address_mask<3:0>	0000
1	Address_mask<7:4>	0000
2	Interrupt - nPfiq	xxx0
3	Interrupt - nPirq	xxx0
4	Interrupt - nFiq2	xxx0
5	Interrupt - nIrq2	xxx0
6	Control - {HaltP2, resetP2}	xx01

Sel	Register set on Read	Default
0..5	Interrupt - {nIrq2, nFiq2, nPirq, nPfiq}	0000
6	Control - {ID1 ,ID0, HaltP2, resetP2}	$d_1d_0$01 where d_1d_0, is the two bit ID field
7	Test - {nFiql, nIrq1}	0001 for test purposes only

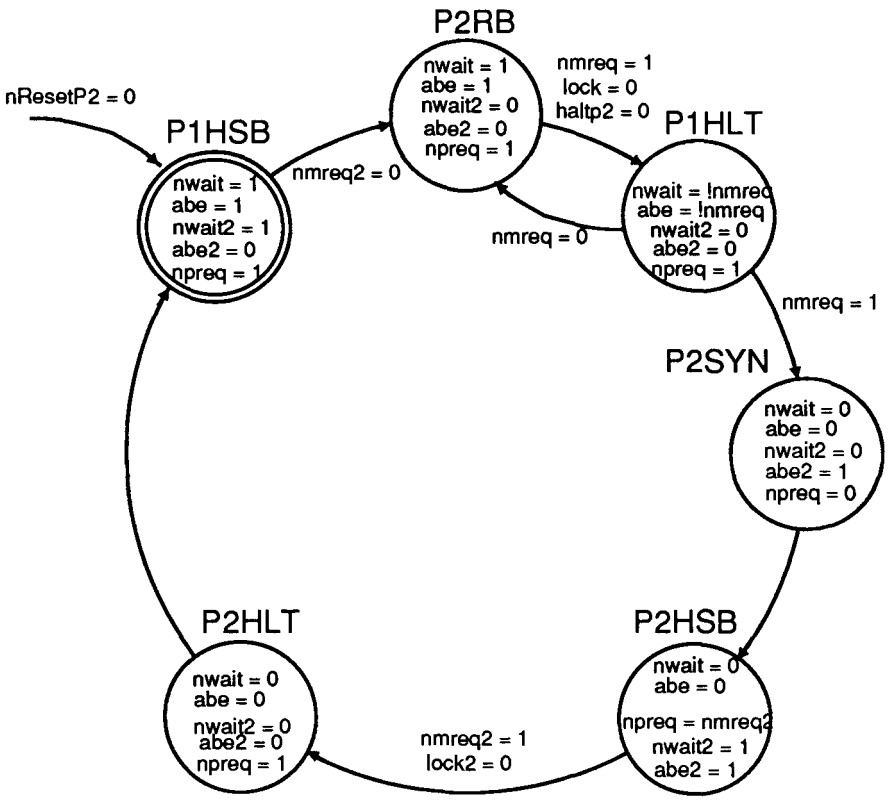
Notes:

The interrupt flag registers drive output signals connected to the 2nd processor, and via the PROBUS connector the podule interrupt lines. Writing a 1 to these registers will therefore force either a normal or fast interrupt to the 1st or second processor. Writing a 0 will remove the interrupt and therefore clears the interrupt condition, this should be done by the interrupt handling routine. Reading the interrupt register will yield a 4 bit value which is all the interrupt registers concatenated and allows the interrupt handler to ascertain what form of interrupt is required.

The podule interrupt lines; nPirq and nPfiq are open collector outputs, and therefore must be pulled high for proper operation.

The Test' register is provided so that the first processor interrupt lines may be monitered on the second CPU card, to ensure they are correctly connected. An alteration to the 'Control Register' PLD would allow these lines to be routed through to the 2nd CPU allowing the 2nd CPU to take over operation of the whole machine.

3. Control State Machine



States

P1HSB Processor 1 HaS control of the Bus
P2RB Processor 2 Requests the Bus and is halted
PIHLT Processor 1 HaLT - remove processor 1 from the bus
P2SYN Processor 2 SYNc - place processor 2 on the bus and request control from 1OMD
P2HSB Processor 2 Has control of the Bus
P2HLT Processor 2 HaLT - remove processor 2 from the bus

4. Verilog Source Listings

```
// Verilog HDL for Duet, duettop _functional

module duettop(mclk, nreset, nR_W, 1DO, 1D1, nmreq, npreq, abe2, nFiql, nIrq1, Reg_Address,
Address_bus, Address_bit_0, P2_Address_bus, Data_Bus, nPfiq, nPirq, nFiq2, nIrq2, nResetP2,
HaltP2, nReg_Out_En, Reg_Strobe);

input mclk, nreset, nR_W, 1DO, 1D1, nmreq, npreq, abe2, nFiql, nlrql;
input [2:0] Reg_Address;
input Address_bit_0;
inout [7:0] Address_bus;
inout [3:0] Data_Bus;
input [7:0] P2_Address_bus;

output nPfiq, nPirq, nFiq2, nIrq2, nResetP2, HaltP2, nReg_Out_En, Reg_Strobe;

wire [7:0] P2_address_mask;

Address_Decode decoder(mclk, {address_bus[7:0] , Address_bit_0}, nR_W, nmreq, npreq,
nReg_Out_En, Reg_Strobe);
control_reg reg_block(nreset, Reg_Address, Data_Bus, nReg_Out_En, Reg_Strobe,ID0, ID1, nFiql,
nlrql, nPfiq, nPirq, nFiq2, nIrq2, HaltP2, nResetP2, P2_address_mask);
address_map mapper(P2_Address_bus, P2_address_mask, abe2, Address_bus);

endmodule


// Verilog HDL for Duet, Address_Decode _functional

module Address_Decode(mclk, Address_bus, nR_W, nmreq, npreq, nReg_Out_En, Reg_Strobe);
        input mclk, nR_W, nmreq, npreq;
        input [8:0] Address_bus;
        output nReg_Out_En, Reg_Strobe;

        wire nReg_Out_En, Reg_Strobe;
        reg Active_Range;

        always @ (negedge mclk)
            Active_Range = (Address_bus[8:0] == 9'h37) & ((nmreq == 0)|(npreq == 0));

        assign nReg_Out_En = !(Active_Range & !nR_W);

        assign Reg_Strobe = !(mclk ? (Active_Range & nR_W) : 0);

endmodule
```

```
// Verilog HDL for Duet, address_map _functional


module address_map(P2_Address_bus, P2_address_mask, abe2, Address_bus);

input [7:0] P2_Address_bus, P2_address_mask;
input abe2;
output [7:0] Address_bus;

wire [7:0] P2_Address_bus, P2_address_mask, Address_bus, mod_P2_Address;

// Apply boot address mask to P2 address bus
        assign mod_P2_Address = P2_Address_bus | P2_address_mask;
        assign Address_bus = abe2? mod_P2_Address: 8'bz;


endmodule
```

```
// Verilog HDL for Duet, control_reg_functional
// Altered register addresses and inverted sense of interrupt reg's 24/2/94
// Altered address decode to avoid fedback register outputs

module control_reg(nreset, Reg_Address, Data_Bus, nReg_Out_En, Reg_Strobe, ID0, ID1, nFiql,
nIrq1, nPfiq, nPirq, nFiq2, nIrq2, HaltP2, nResetP2, Boot_Add_Mask);


        input nreset, nReg_Out_En, Reg_Strobe, 1D0, ID1, nFiql, nIrql;
        input [2:0] Reg_Address;
        inout [3:0] Data_Bus;

        output nPfiq, nPirq, nFiq2, nIrq2, HaltP2,nResetP2;
        output [7:0] Boot_Add_Mask;

        reg [7:0] Boot_Add_Mask;
        reg [3:0] Interrupt;
        reg [1:0] CntIP2;

        wire [3:0] dataout, Data_Bus;

 // reset registers to appropriate values
        always @ (nreset)
            if (nreset == 0)
              begin
                  assign Boot_Add_Mask = 0;
                  assign Interrupt =4'b0000;
                  assign CntIP2 = 2'b01;
              end
             else
              begin
                  deassign Boot_Add_Mask;
                  deassign Interrupt;
                  deassign CntIP2;
               end

 // write data into register block
          always @(posedge Reg_Strobe)
              case (Reg_Address)
                    3'h0: Boot_Add_Mask[3:0] = Data_Bus;
                    3'h1: Boot_Add_Mask[7:4] = Data_Bus;
                    3'h2: Interrupt[0]  = Data_Bus[0];
                    3'h3: Interrupt[1]  = Data_Bus[0];
                    3'h4: Interrupt[2]  = Data_Bus[0];
                    3'h5: Interrupt[3]  = Data_Bus [0];
                    3'h6: CntIP2        = Data_Bus[1:0];
                    default:;
              endcase
```

```
// assign register values to output pins
// Interrupt lines must be open collector !
        assign nPfiq = Interrupt[0]? 0 : l'bz;
        assign nPirq = Interrupt[1]? 0 : l'bz;
        assign nFiq2 = !Interrupt[2];
        assign nIrq2 = !Interrupt[3];

        assign nResetP2 = !((nreset = 0) | (Cnt1P2[0] = 1));
        assign HaltP2 = CntIP2[1];

// place all readable registers on dataout bus
        assign dataout[3:0] = (Reg_Address == 6)? {id1 ,ID0,Cnt1P2[1:0] } :
                              (Reg_Address == 7)? { 1 'b0,1'b0,nFiql,nIrq1 } :
                                    Interrupt[3:0];

// gate dataout bus onto databus under control of Reg_Out_En
        assign Data_Bus = nReg_Out_En? 4'bz : dataout;

endmodule
```

```
module duet(mclk, nResetP2, HaltP2, nmreq, nmreq2, lock, lock2, npreq, abe, abe2, nwait, nwait2);

  input mclk, nResetP2, HaltP2, nmreq, nmreq2, lock, lock2;
  output npreq, abe, abe2, nwait, nwait2;

  reg [3:0] state;

'define PIHSB 3'b000 '
define P2RQB 3b001 '
define P1HLT 3'b010 '
define P2SYN 3b011 '
define P2HSB 3'b100 '
define P2HLT 3'b101


// Decode outputs from state machine current state & two async control signals
        assign nwait = (state == 'P1HSB) | (state == 'P2RQB) | ((state == 'P1HLT) & !nmreq);
        // Trap single idle
        assign abe = (state == 'P1HSB) | (state == 'P2RQB) | ((state == 'P1HLT) & !nmreq); // cycles
        assign nwait2       = (state == 'P1HSB) | (state== 'P2HSB);
        assign abe2 = (state == 'P2SYN) | (state == 'P2HSB);
        assign npreq = (state != 'P2SYN) & ((state != 'P2HSB) | nmreq2);
// To ensure IOMD sees only 1 cycle of valid 'nmreq2'

// Reset condition
        always @(nResetP2)
            if (nResetP2 = 0)
              assign state = 'P1HSB;
            else
              deassign state;

// Define state sequence
        always @ (negedge mclk)
        begin
            case (state)
                'P1HSB: if (nmreq2 == 0) state = 'P2RQB;
                'P2RQB: if ((nmreq == 1) & (lock == 0) & (HaltP2 == 0)) state = 'P1HLT;
                'P1HLT: if (nmreq == 1) state = 'P2SYN;
                        else state = 'P2RQB;
                 'P2SYN: state = 'P2HSB;
                'P2HSB: if ((nmreq2 == 1) & (lock2 == 0)) state = 'P2HLT;
                'P2I-ILT: state = 'P1HSB;
                default: state = 'P1HSB;
              endcase
        end

endmodule
```