



INT selection

DIP[6,7] selects which DSP interrupt is generated at the end of a generic ADC conversion.

DIP[7]	DIP[6]	INT selection	Mode
0	0	INT0	Active High
0	1	INT1	Active High
1	0	INT2	Active Low
1	1	INT3	Active Low

Table 2: DSP Interrupt selection switches

INT enable

DIP[8] enable or disable the interrupt generation

DIP[8]	Enable/Disable
0	INT disabled
1	INT enable

Table 3: INT enable switch

Automatic conversion configuration selection

DIP[9,10] selects how to use the on-board ADC and DAC converters.

DIP[10]	DIP[9]	ADC Start of conversion signal	DAC load signal
0	0	Software control only	Software control only
0	1	TCLK0 only	TCLK0 or software control
1	0	TCLK1 only	TCLK1 or software control
1	1	RESERVED	RESERVED

Table 4: Automatic conversion configuration selection



1. Software control

- to begin DAC conversions a 0x01 must be written by DSP into SOC REG register (See memory map, table 6)
- to begin ADC conversions a 0x02 must be written by DSP into SOC REG register (See memory map, table 6)
- to begin both ADC and DAC conversions a 0x03 must be written by DSP into SOC REG register (See memory map, table 6)

2. TCLK0: both ADC and DAC conversions start when DSP timer TCLK0 signal go high

3. TCLK1: both ADC and DAC conversions start when DSP timer TCLK1 signal go high

Enable PC access

DIP[11] enables access to the I/O board from an external host computer using EPP port or any other communication port.

DIP[11]	Enable/Disable
0	Disable PC access
1	Enable PC access

Table 5: Enable PC access



I/O Board Memory Map Description

Table 6 shows the memory map of the I/O board.

HEX	USE	R/W	COMMENTS
BASE+00	DAC CH0	W	The 4 most significant bits are ignored
BASE+01	DAC CH1	W	The 4 most significant bits are ignored
BASE+02	DAC CH2	W	The 4 most significant bits are ignored
BASE+03	DAC CH3	W	The 4 most significant bits are ignored
BASE+04	DAC CH4	W	The 4 most significant bits are ignored
BASE+05	DAC CH5	W	The 4 most significant bits are ignored
BASE+06	DAC CH6	W	The 4 most significant bits are ignored
BASE+07	DAC CH7	W	The 4 most significant bits are ignored
BASE+00	ADC REG0	R	The 4 most significant bits are set to 0
BASE+01	ADC REG1	R	The 4 most significant bits are set to 0
BASE+02	ADC REG2	R	The 4 most significant bits are set to 0
BASE+03	ADC REG3	R	The 4 most significant bits are set to 0
BASE+04	ADC REG4	R	The 4 most significant bits are set to 0
BASE+05	ADC REG5	R	The 4 most significant bits are set to 0
BASE+06	ADC REG6	R	The 4 most significant bits are set to 0
BASE+07	ADC REG7	R	The 4 most significant bits are set to 0
BASE+08	DIGITAL IN[0..15]	R	--
BASE+09	DIGITAL IN[16..31]	R	--
BASE+08	DIGITAL OUT[0..15]	W	--
BASE+09	DIGITAL OUT[16..31]	W	--
BASE+0A	ENABLE REG	R W	The 8 most significant bits are set to 0 The 8 most significant bits are ignored
BASE+0B	CONFIG REG	R/W	ADC Configuration Register: BIP REG (8 bit) and RNG REG (8 bit)
BASE+0C	EOC REG	R	The 15 most significant bits are set to 0
BASE+0C	SOC REG	W	The 14 most significant bits are ignored

Table 6: memory map of the I/O board



- **DAC CHn (write):** this register represents a DAC channel n: when writing a value in this register, the 4 most significant bits are ignored (DAC are 12 bits converters).

X	X	X	X	MSB	MSB-1	LSB+1	LSB
---	---	---	---	-----	-------	-----	-----	-----	-----	-----	-----	-----	-----	-------	-----

- **ADC REGn (read):** this register represents an ADC channel n: when reading a value from this register, the 4 most significant bits are always set to zero (ADC are 12 bits converters)

0	0	0	0	MSB	MSB-1	LSB+1	LSB
---	---	---	---	-----	-------	-----	-----	-----	-----	-----	-----	-----	-----	-------	-----

- **DIGITAL IN (read):** these two 16 bit registers corresponds to the 32 digital input channels available on the DIGITAL IN connector of the I/O board. The DSP can access to these channels by reading BASE+08 and BASE+09 addresses.

BASE+08	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
---------	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

BASE+09	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
---------	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

- **DIGITAL OUT (write):** these two 16 bit registers corresponds to the 32 digital output channels available on the DIGITAL OUT connector of the I/O board. The DSP can access to these channels by writing BASE+08 and BASE+09 addresses.

BASE+08	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
---------	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

BASE+09	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
---------	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

- **ENABLE REG (read/write):** this register allows enabling or disabling each ADC channels independently. Each of the 8 least significant bits of the ENABLE REG represents one of the 8 ADC channel:
 - if the bit is set to 1, the corresponding ADC channel is enabled
 - if the bit is set to 0, the corresponding ADC channel is disabled



When writing (to set the configuration):

X	X	X	X	X	X	X	X	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
---	---	---	---	---	---	---	---	------	------	------	------	------	------	------	------

When reading (to check the configuration):

0	0	0	0	0	0	0	0	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
---	---	---	---	---	---	---	---	------	------	------	------	------	------	------	------

ADCn = 1 ADC channel n enabled
ADCn = 0 ADC channel n disabled

- **CONFIG REG (read/write):** this register contains two 8-bit registers:
 - 8 least significant bits → RNG register
 - 8 most significant bits → BIP register

Each bit of these registers represents a single ADC channel:

- when a bit of RNG register is set to 1, the correspondent ADC max. input value is set to 10 volts
- when a bit of RNG register is set to 0, the correspondent ADC max. input value is set to 5 volts
- when a bit of BIP register is set to 1, the correspondent ADC input range is bipolar
- when a bit of BIP register is set to 0, the correspondent ADC input range is unipolar

ADC 7	ADC 6	ADC 5	ADC 4	ADC 3	ADC 2	ADC 1	ADC 0	ADC 7	ADC 6	ADC 5	ADC 4	ADC 3	ADC 2	ADC 1	ADC 0
{ BIP register – polarity of analog input signals }								{ RNG register – range of analog input signals }							



Example of CONFIG REG:

0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADC0: 10 volts, bipolar → analog input range $-10 \div +10$ volts
ADC1: 5 volts, bipolar → analog input range $-5 \div +5$ volts
ADC2 ÷ ADC7: 5 volts, unipolar → analog input range $0 \div +5$ volts

- **EOC REG (read):**

- the bit is set to 1 at the end of a generic ADC conversion
- the bit is set to 0 during any other operation of ADC converters

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EOC REG
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---------

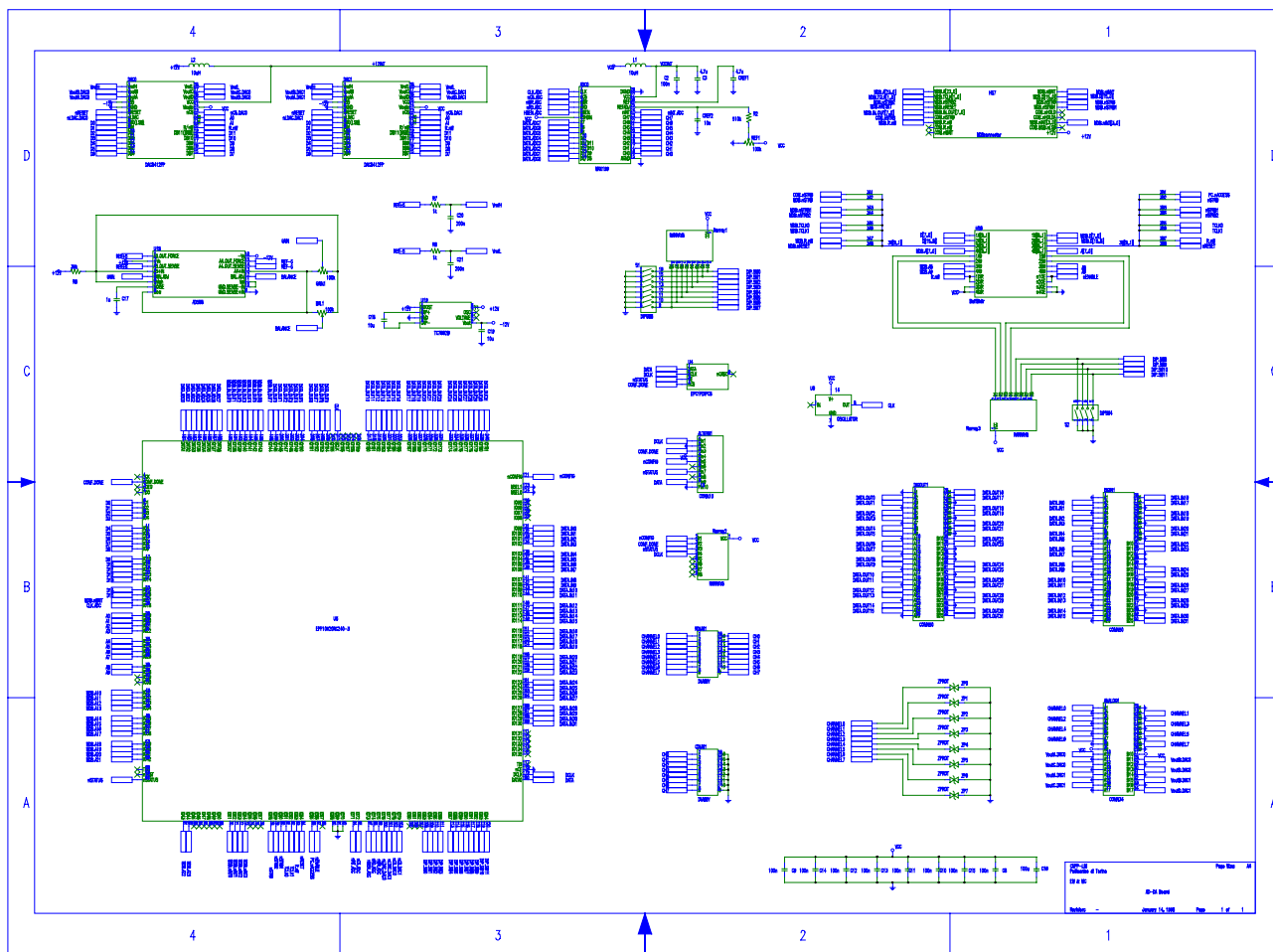
- **SOC REG (write):** this register is used to generate a software start of conversion:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	SOC REG MSB	SOC REG LSB
---	---	---	---	---	---	---	---	---	---	---	---	---	---	----------------	----------------

- if the LSB is set to 1, a software DAC start of conversion is generated.
- if the MSB is set to 1, a software ADC start of conversion is generated.



I/O board schematic diagrams





ALTERA schematic diagrams



ALTERA VHDL code: AUTOMAT.VHD

```
-----  
-- Standard Library calls  
-----  
LIBRARY ieee;  
USE ieee.STD_LOGIC_1164.all;  
USE ieee.STD_LOGIC_arith.all;  
  
-----  
-- Entity Section  
-----  
ENTITY AUTOMAT IS  
  PORT  
  (  
    -- General inputs/outputs  
    DIP_SW_RAM : IN STD_LOGIC_VECTOR(5 downto 0); -- Dip Switches used to  
                                                    -- define address where  
                                                    -- I/O is mapped  
    DIP_SW_AUTO_CONV_CONFIG : IN STD_LOGIC_VECTOR(1 downto 0); -- If '1' ADC is  
                                                    -- controlled  
                                                    -- automatically  
    DIP_SW_INTERRUPT_ENABLE : IN STD_LOGIC; -- If '1' ADC generates an  
                                                    -- interrupt on DSP  
    DIP_SW_ALLOW_PC_ACCESS : IN STD_LOGIC; -- If '1' allows access to  
                                                    -- I/O from host PC  
                                                    -- (through EPP protocol)  
    CLK : IN STD_LOGIC; -- I/O internal clock  
                                                    -- signal  
    ENABLE_ALTERA_TRISTATE_OUTPUTS : OUT STD_LOGIC; -- Signal that enables  
                                                    -- internal ALTERA tri-  
                                                    -- state buffers  
    nENABLE : OUT STD_LOGIC; -- External bi-directional  
                                -- driver enable signal  
    -- Open-DSP Bus signals  
    nRESET : IN STD_LOGIC; -- Main reset (low active)  
    INTERNAL_nSTRB : IN STD_LOGIC; -- Strobe signal (selected  
                                -- on main ALTERA  
                                -- schematics  
                                -- 'SCHEDAIO.GDF')  
    PC_nACCESS : IN STD_LOGIC; -- If '1' a DSP access is  
                                -- performed, If '0' a PC  
                                -- access is performed  
    R_nW : IN STD_LOGIC; -- If '1' a read cycle is  
                                -- performed, if '0' a  
                                -- write cycle is  
                                -- performed  
    TCLK0 : IN STD_LOGIC; -- DSP timer0 signal  
    TCLK1 : IN STD_LOGIC; -- DSP timer1 signal  
    D_IN : IN STD_LOGIC_VECTOR(15 downto 0); -- Input data bus  
    D_OUT : OUT STD_LOGIC_VECTOR(15 downto 0); -- Output data bus  
    A : IN STD_LOGIC_VECTOR(9 downto 0); -- Address bus  
    nINT : OUT STD_LOGIC; -- Interrupt signal to be  
                                -- send to one of DSP  
                                -- interrupts  
  )  
END ENTITY AUTOMAT;
```



Center for Prototyping Services

Mechatronics Laboratory



```
-- Digital I/O signals
DIGITAL_IN0      : IN  STD_LOGIC_VECTOR(15 downto 0); -- Digital input from
                                                         connector DIGIN0
DIGITAL_IN1      : IN  STD_LOGIC_VECTOR(15 downto 0); -- Digital input from
                                                         connector DIGIN1
DIGITAL_OUT0     : OUT STD_LOGIC_VECTOR(15 downto 0); -- Digital output to
                                                         connector DIGOUT0
DIGITAL_OUT1     : OUT STD_LOGIC_VECTOR(15 downto 0); -- Digital output to
                                                         connector DIGOUT1

-- DAC signals
nCS_DAC0         : OUT STD_LOGIC; -- DAC 0 enable signal
                                                         (low active)
nCS_DAC1         : OUT STD_LOGIC; -- DAC 1 enable signal
                                                         (low active)
nLDAC_DAC0       : OUT STD_LOGIC; -- DAC 0 load latch signal
                                                         (low active)
nLDAC_DAC1       : OUT STD_LOGIC; -- DAC 1 load latch signal
                                                         (low active)

-- ADC signals
DATA_ADC_IN      : IN  STD_LOGIC_VECTOR(7 downto 0); -- Input data bus from ADC
DATA_ADC_OUT     : OUT STD_LOGIC_VECTOR(7 downto 0); -- Output data bus to ADC
nINT_ADC         : IN  STD_LOGIC; -- Interrupt signal from
                                                         ADC
nCS_ADC          : OUT STD_LOGIC; -- ADC enable signal (low
                                                         active)
nRD_ADC          : OUT STD_LOGIC; -- ADC read signal (low
                                                         active)
nWR_ADC          : OUT STD_LOGIC; -- ADC write signal (low
                                                         active)
HBEN_ADC         : OUT STD_LOGIC -- ADC high/low data read
                                                         selector. If '1' high
                                                         four bits are read, if
                                                         '0' low 8 bits are read

);
END AUTOMAT;

-- End of Entity Section
```



Center for Prototyping Services

Mechatronics Laboratory



-- Architecture Section

ARCHITECTURE a OF AUTOMAT IS

-- Architecture Signals

-- General Signals

SIGNAL ENABLE_SIGNAL : STD_LOGIC;

-- If '1' a Open-DSP to I/O board is in progress, if '0' Open-DSP has not selected the I/O board

-- Conversion Registers

SIGNAL CONVERSION_STATE : STD_LOGIC_VECTOR(2 downto 0);

-- State of automatic ADC conversion control

SIGNAL CONVERTER_CHANNEL : STD_LOGIC_VECTOR(2 downto 0);

-- Analog channel that is selected from ADC

SIGNAL START_OF_CONVERSION : STD_LOGIC;

-- If '1' an ADC conversion will begin

SIGNAL START_OF_CONVERSION_SOFT : STD_LOGIC;

-- If '1' a software controlled ADC conversion will begin

SIGNAL START_OF_CONVERSION_TCLK0 : STD_LOGIC;

-- If '1' an ADC conversion will begin every time that DSP timer 0 signal go high

SIGNAL START_OF_CONVERSION_TCLK1 : STD_LOGIC;

-- If '1' an ADC conversion will begin every time that DSP timer 1 signal go high

SIGNAL END_OF_CONVERSION : STD_LOGIC;

-- After a conversion is finished, this signal go high

SIGNAL DELAY_END_OF_CONVERSION : STD_LOGIC;

-- Auxiliary end of conversion signal

SIGNAL INTERNAL_nINT : STD_LOGIC;

-- Internal signal mirror of nINT signal

SIGNAL RESET_nINT_MDB : STD_LOGIC;

-- Internal signal that is active when nRESET is '0' or when a start of conversion is received

SIGNAL nLDAC_DAC_TCLK0 : STD_LOGIC;

-- If '0' a DAC conversion will begin every time that DSP timer 0 signal go high

SIGNAL nLDAC_DAC_TCLK1 : STD_LOGIC;

-- If '0' a DAC conversion will begin every time that DSP timer 1 signal go high

SIGNAL nLDAC_DAC_SOFT : STD_LOGIC;

-- If '0' a software controlled DAC conversion will begin

SIGNAL nLDAC_DAC_SIGNAL : STD_LOGIC;

-- If '0' a DAC conversion will begin

SIGNAL LDAC_DAC_COUNTER : STD_LOGIC_VECTOR(1 downto 0);

-- A counter that control the DAC load minimum time

SIGNAL LDAC_DAC_EOL : STD_LOGIC;

-- Signal that indicates the DAC End Of Load



Center for Prototyping Services

Mechatronics Laboratory



```
-- ADC Registers
SIGNAL ADC_REG                                : STD_LOGIC_VECTOR(95 downto 0);

SIGNAL RNG_REG                                : STD_LOGIC_VECTOR(7 downto 0);

SIGNAL BIP_REG                                : STD_LOGIC_VECTOR(7 downto 0);

SIGNAL ENABLE_REG                             : STD_LOGIC_VECTOR(7 downto 0);

-- Digital I/O Registers
SIGNAL REG0                                    : STD_LOGIC_VECTOR(15 downto 0);

SIGNAL REG1                                    : STD_LOGIC_VECTOR(15 downto 0);
```

-- 8 Internal registers each of 12 bits that stores the last ADC conversion data for every ADC channel

-- A register of 8 bits where each bit represents the voltage range of each ADC channel. If '1' a range of 10V is selected, if '0' a range of 5V is selected

-- A register of 8 bits where each bit represents the voltage polarity of each ADC channel. If '1' the input can be positive or negative voltage, if '0' the input voltage is positive only

-- A register of 8 bits where each bit enable an ADC channel. if '1' the channel is enabled, if '0' the channel is disabled

-- A register of 16 bit that stores the least significative bits of digital output (DIGIOUT)

-- A register of 16 bit that stores the most significative bits of digital output (DIGIOUT)



-- Architecture Processes Description

BEGIN

-- Generate "enable signals" for Open-DSP read or write access cycles
-- Signals generated into process:
-- nCS_DAC0, nCS_DAC1, ENABLE_ALTERA_TRISTATE_OUTPUTS, ENABLE_SIGNAL, nINT, RESET_nINT_MDB,
-- nLDAC_DAC_SIGNAL, nLDAC_DAC0, nLDAC_DAC1, DIGITAL_OUT0, DIGITAL_OUT1, nENABLE

```
PROCESS
BEGIN
    IF (nRESET = '0') THEN
        nCS_DAC0 <= '1';
        nCS_DAC1 <= '1';
        ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';
    ELSIF (ENABLE_SIGNAL = '1') THEN
        CASE A(3 downto 0) IS
            -- DAC chips (Write only) or ADC Conversion Registers (Read only)
            WHEN "0000" | "0001" | "0010" | "0011" =>
                IF (R_nW = '0') THEN
                    nCS_DAC0 <= INTERNAL_nSTRB;
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';
                ELSE
                    nCS_DAC0 <= '1';
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= NOT(INTERNAL_nSTRB);
                END IF;
                nCS_DAC1 <= '1';
            WHEN "0100" | "0101" | "0110" | "0111" =>
                IF (R_nW = '0') THEN
                    nCS_DAC1 <= INTERNAL_nSTRB;
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';
                ELSE
                    nCS_DAC1 <= '1';
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= NOT(INTERNAL_nSTRB);
                END IF;
                nCS_DAC0 <= '1';
            -- ADC configuration registers
            WHEN "1010" | "1011" =>
                IF (R_nW = '1') THEN
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= NOT(INTERNAL_nSTRB);
                ELSE
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';
                END IF;
                nCS_DAC0 <= '1';
                nCS_DAC1 <= '1';
            -- Control of Conversion register
            WHEN "1100" =>
                IF (R_nW = '1') THEN
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= NOT(INTERNAL_nSTRB);
                ELSE
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';
                END IF;
                nCS_DAC0 <= '1';
                nCS_DAC1 <= '1';
            -- Digital input locations
            WHEN "1000" | "1001" =>
                IF (R_nW = '1') THEN
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= NOT(INTERNAL_nSTRB);
                ELSE
                    ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';
                END IF;
                nCS_DAC0 <= '1';
                nCS_DAC1 <= '1';
```



Center for Prototyping Services

Mechatronics Laboratory



```
        WHEN OTHERS =>
            nCS_DAC0 <= '1';
            nCS_DAC1 <= '1';
            ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';

        END CASE;

    ELSE
        nCS_DAC0 <= '1';
        nCS_DAC1 <= '1';
        ENABLE_ALTERA_TRISTATE_OUTPUTS <= '0';

    END IF;

    IF (A(9 downto 4) = DIP_SW_RAM) THEN
        ENABLE_SIGNAL <= PC_nACCESS OR DIP_SW_ALLOW_PC_ACCESS;

    ELSE
        ENABLE_SIGNAL <= '0';

    END IF;

    IF (DIP_SW_INTERRUPT_ENABLE = '1') THEN
        nINT <= INTERNAL_nINT;

    ELSE
        nINT <= '1';

    END IF;

    RESET_nINT_MDB <= START_OF_CONVERSION OR NOT(nRESET);
    nLDAC_DAC_SIGNAL <= nLDAC_DAC_TCLK0 AND nLDAC_DAC_TCLK1 AND nLDAC_DAC_SOFT;
    nLDAC_DAC0 <= nLDAC_DAC_SIGNAL;
    nLDAC_DAC1 <= nLDAC_DAC_SIGNAL;
    DIGITAL_OUT0 <= REG0;
    DIGITAL_OUT1 <= REG1;
    nENABLE <= NOT(ENABLE_SIGNAL) OR INTERNAL_nSTRB;
END PROCESS;
```



Center for Prototyping Services

Mechatronics Laboratory



```
-- Open-DSP Read from Internal Registers and Digital Inputs. If data read has less than 16 bit,  
-- zeros are generated for non-used bits  
-- Signals generated into process:  
-- D_OUT
```

```
PROCESS  
BEGIN  
    IF (INTERNAL_nSTRB = '0') THEN  
        IF (ENABLE_SIGNAL = '1' AND R_nW = '1') THEN  
            CASE A(3 downto 0) IS  
                -- ADC conversion registers (read only)  
                WHEN "0000" =>  
                    D_OUT(11 downto 0) <= ADC_REG(11 downto 0);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0001" =>  
                    D_OUT(11 downto 0) <= ADC_REG(23 downto 12);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0010" =>  
                    D_OUT(11 downto 0) <= ADC_REG(35 downto 24);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0011" =>  
                    D_OUT(11 downto 0) <= ADC_REG(47 downto 36);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0100" =>  
                    D_OUT(11 downto 0) <= ADC_REG(59 downto 48);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0101" =>  
                    D_OUT(11 downto 0) <= ADC_REG(71 downto 60);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0110" =>  
                    D_OUT(11 downto 0) <= ADC_REG(83 downto 72);  
                    D_OUT(15 downto 12) <= "0000";  
                WHEN "0111" =>  
                    D_OUT(11 downto 0) <= ADC_REG(95 downto 84);  
                    D_OUT(15 downto 12) <= "0000";  
                -- ADC configuration registers: ENABLE_REG and RNG_REG  
                WHEN "1010" =>  
                    D_OUT(7 downto 0) <= ENABLE_REG;  
                    D_OUT(15 downto 8) <= "00000000";  
                WHEN "1011" =>  
                    D_OUT(7 downto 0) <= RNG_REG;  
                    D_OUT(15 downto 8) <= BIP_REG;  
                -- Control of Conversion register  
                WHEN "1100" =>  
                    D_OUT(0) <= NOT(INTERNAL_nINT);  
                    D_OUT(15 downto 1) <= "0000000000000000";  
                -- Digital input locations  
                WHEN "1000" =>  
                    D_OUT <= DIGITAL_IN0;  
                WHEN "1001" =>  
                    D_OUT <= DIGITAL_IN1;  
                WHEN OTHERS =>  
                    END CASE;  
            END IF;  
        END IF;  
    END PROCESS;
```



Center for Prototyping Services

Mechatronics Laboratory



-- Open-DSP Write to Internal Registers

-- Signals generated into process:

-- RNG_REG, BIP_REG, ENABLE_REG, REG0, REG1

```
PROCESS (INTERNAL_nSTRB, nRESET)
BEGIN
    IF (nRESET = '0') THEN
        RNG_REG <= "11111111";
        BIP_REG <= "11111111";
        ENABLE_REG <= "00000000";
        REG0 <= "0000000000000000";
        REG1 <= "0000000000000000";
    ELSIF (INTERNAL_nSTRB'EVENT AND INTERNAL_nSTRB = '1') THEN
        IF (ENABLE_SIGNAL = '1' AND R_nW = '0') THEN
            CASE A(3 downto 0) IS
                -- ADC configuration registers: ENABLE_REG and RNG_REG
                WHEN "1010" =>
                    ENABLE_REG <= D_IN(7 downto 0);
                WHEN "1011" =>
                    RNG_REG <= D_IN(7 downto 0);
                    BIP_REG <= D_IN(15 downto 8);
                -- Digital output registers locations
                WHEN "1000" =>
                    REG0 <= D_IN;
                WHEN "1001" =>
                    REG1 <= D_IN;
                WHEN OTHERS =>
                    END CASE;
            END IF;
        END IF;
    END PROCESS;
```

-- Generate final ADC start of conversion signal from individual ADC start of conversion signals

-- Signals generated into process:

-- START_OF_CONVERSION

```
PROCESS (CLK, nRESET)
BEGIN
    IF nRESET = '0' THEN
        START_OF_CONVERSION <= '0';
    ELSIF (CLK'EVENT AND CLK = '1') THEN
        START_OF_CONVERSION <= (START_OF_CONVERSION_TCLK0 OR
START_OF_CONVERSION_TCLK1
                                OR START_OF_CONVERSION_SOFT);
    END IF;
END PROCESS;
```

-- Generate DAC end of load signal

-- Signals generated into process:

-- LDAC_DAC_EOL

```
PROCESS (CLK, nRESET)
BEGIN
    IF nRESET = '0' THEN
        LDAC_DAC_COUNTER <= "00";
    ELSIF (CLK'EVENT AND CLK = '1') THEN
        IF (nLDAC_DAC_SIGNAL = '0') THEN
            LDAC_DAC_COUNTER <= UNSIGNED(LDAC_DAC_COUNTER) + 1;
        ELSE
            LDAC_DAC_COUNTER <= "00";
        END IF;
    END IF;
    LDAC_DAC_EOL <= LDAC_DAC_COUNTER(1);
```



Center for Prototyping Services

Mechatronics Laboratory



END PROCESS;

-- Generate load DAC signal every time that TCLK0 go high
-- Signals generated into process:
-- nLDAC_DAC_TCLK0

PROCESS (TCLK0, LDAC_DAC_EOL)
BEGIN

IF LDAC_DAC_EOL = '1' THEN

nLDAC_DAC_TCLK0 <= '1';

ELSIF (TCLK0'EVENT AND TCLK0 = '1') THEN

IF (DIP_SW_AUTO_CONV_CONFIG = "01") THEN -- Begin of Conversion is controlled

by TCLK0

nLDAC_DAC_TCLK0 <= '0';

ELSE

nLDAC_DAC_TCLK0 <= '1';

END IF;

END IF;

END PROCESS;

-- Generate load DAC signal every time that TCLK1 go high
-- Signals generated into process:
-- nLDAC_DAC_TCLK1

PROCESS (TCLK1, LDAC_DAC_EOL)
BEGIN

IF LDAC_DAC_EOL = '1' THEN

nLDAC_DAC_TCLK1 <= '1';

ELSIF (TCLK1'EVENT AND TCLK1 = '1') THEN

IF (DIP_SW_AUTO_CONV_CONFIG = "10") THEN -- Begin of Conversion is controlled

by TCLK1

nLDAC_DAC_TCLK1 <= '0';

ELSE

nLDAC_DAC_TCLK1 <= '1';

END IF;

END IF;

END PROCESS;

-- Generate load DAC signal every time that DSP writes a software command
-- Signals generated into process:
-- nLDAC_DAC_SOFT

PROCESS (INTERNAL_nSTRB, LDAC_DAC_EOL)
BEGIN

IF LDAC_DAC_EOL = '1' THEN

nLDAC_DAC_SOFT <= '1';

ELSIF (INTERNAL_nSTRB'EVENT AND INTERNAL_nSTRB = '1') THEN

IF (ENABLE_SIGNAL = '1' AND A(3 downto 0) = "1100") THEN -- Control of

Conversion register

IF (R_nW = '0') THEN -- ADC Conversions are started by software

command

nLDAC_DAC_SOFT <= NOT(D_IN(0));

ELSE

nLDAC_DAC_SOFT <= '1';

END IF;

END IF;

END IF;

END PROCESS;



Center for Prototyping Services

Mechatronics Laboratory



```
-- Generate internal interrupt signal when End of Conversion is detected. Interrupt is cleared
-- when Start of Conversion is enabled
-- Signals generated into process:
-- INTERNAL_nINT
```

```
PROCESS (END_OF_CONVERSION, RESET_nINT_MDB)
BEGIN
    IF RESET_nINT_MDB = '1' THEN
        INTERNAL_nINT <= '1';
    ELSIF (END_OF_CONVERSION'EVENT AND END_OF_CONVERSION = '0') THEN
        INTERNAL_nINT <= '0';
    END IF;
END PROCESS;
```

```
-- Generate ADC Start of Conversion every time that TCLK0 go high
-- Signals generated into process:
-- START_OF_CONVERSION_TCLK0
```

```
PROCESS (TCLK0, END_OF_CONVERSION)
BEGIN
    IF END_OF_CONVERSION = '1' THEN
        START_OF_CONVERSION_TCLK0 <= '0';
    ELSIF (TCLK0'EVENT AND TCLK0 = '1') THEN
        IF (DIP_SW_AUTO_CONV_CONFIG = "01") THEN -- Begin of Conversion is controlled
            START_OF_CONVERSION_TCLK0 <= '1';
        ELSE
            START_OF_CONVERSION_TCLK0 <= '0';
        END IF;
    END IF;
END PROCESS;
```

by TCLK0

```
-- Generate ADC Start of Conversion every time that TCLK1 go high
-- Signals generated into process:
-- START_OF_CONVERSION_TCLK1
```

```
PROCESS (TCLK1, END_OF_CONVERSION)
BEGIN
    IF END_OF_CONVERSION = '1' THEN
        START_OF_CONVERSION_TCLK1 <= '0';
    ELSIF (TCLK1'EVENT AND TCLK1 = '1') THEN
        IF (DIP_SW_AUTO_CONV_CONFIG = "10") THEN -- Begin of Conversion is controlled
            START_OF_CONVERSION_TCLK1 <= '1';
        ELSE
            START_OF_CONVERSION_TCLK1 <= '0';
        END IF;
    END IF;
END PROCESS;
```

by TCLK1

Document Author(s): Eduardo Miranda, Marcello Chiaberge

Document Title: I/O Board Documentation

File name: W:\OpenDSP_GeneralPurposeIO.doc

Created: 06-07-98 18:55

Last editing: 10-07-98 17:16

Page 19 of 25



Center for Prototyping Services

Mechatronics Laboratory



```
-- Generate ADC Start of Conversion every time that DSP writes a software command
-- Signals generated into process:
-- START_OF_CONVERSION_SOFT
```

```
PROCESS (INTERNAL_nSTRB, END_OF_CONVERSION)
BEGIN
```

```
    IF END_OF_CONVERSION = '1' THEN
```

```
        START_OF_CONVERSION_SOFT <= '0';
```

```
    ELSIF (INTERNAL_nSTRB'EVENT AND INTERNAL_nSTRB = '1') THEN
```

```
        IF (ENABLE_SIGNAL = '1' AND A(3 downto 0) = "1100") THEN -- Control of
```

Conversion register

```
            IF (DIP_SW_AUTO_CONV_CONFIG = "00" AND R_nW = '0') THEN -- ADC
```

Conversions are started

by software command

```
                START_OF_CONVERSION_SOFT <= D_IN(1);
```

```
            ELSE
```

```
                START_OF_CONVERSION_SOFT <= '0';
```

```
            END IF;
```

```
    END IF;
```

```
END IF;
```

```
END PROCESS;
```

```
-- Finite state machine that controls ADC automatic conversion
```

```
-- Signals generated into process:
```

```
-- CONVERSION_STATE, CONVERTER_CHANNEL, nCS_ADC, nWR_ADC, nRD_ADC, HBEN_ADC,
```

```
-- END_OF_CONVERSION, DELAY_END_OF_CONVERSION, DATA_ADC_OUT, ADC_REG
```

```
PROCESS (CLK, nRESET)
```

```
BEGIN
```

```
    IF (nRESET = '0') THEN
```

```
        CONVERSION_STATE <= "000";
```

```
        CONVERTER_CHANNEL <= "000";
```

```
        nCS_ADC <= '1';
```

```
        nWR_ADC <= '1';
```

```
        nRD_ADC <= '1';
```

```
        HBEN_ADC <= '1';
```

```
        END_OF_CONVERSION <= '0';
```

```
        DELAY_END_OF_CONVERSION <= '0';
```

```
    ELSIF (CLK'EVENT AND CLK = '1') THEN
```

```
        IF (START_OF_CONVERSION = '1' AND END_OF_CONVERSION = '0') THEN -- Begin a
```

conversion

```
            IF (CONVERSION_STATE = "000") THEN -- Write control byte into the
```

ADC converter

```
                END_OF_CONVERSION <= '0';
```

```
                HBEN_ADC <= '1';
```

```
                CASE CONVERTER_CHANNEL IS
```

```
                    WHEN "000" =>
```

```
                        IF (ENABLE_REG(0) = '1') THEN
```

```
                            DATA_ADC_OUT(7 downto 5) <= "000";
```

```
                            DATA_ADC_OUT(4) <= RNG_REG(0);
```

```
                            DATA_ADC_OUT(3) <= BIP_REG(0);
```



Center for Prototyping Services

Mechatronics Laboratory



```
CONVERTER_CHANNEL;

DATA_ADC_OUT(2 downto 0) <=

nCS_ADC <= '0';
nWR_ADC <= '0';
nRD_ADC <= '1';
CONVERSION_STATE <= "001";
ELSE
CONVERSION_STATE <= "101";
END IF;

WHEN "001" =>
IF (ENABLE_REG(1) = '1') THEN
DATA_ADC_OUT(7 downto 5) <= "000";
DATA_ADC_OUT(4) <= RNG_REG(1);
DATA_ADC_OUT(3) <= BIP_REG(1);
DATA_ADC_OUT(2 downto 0) <=

nCS_ADC <= '0';
nWR_ADC <= '0';
nRD_ADC <= '1';
CONVERSION_STATE <= "001";
ELSE
CONVERSION_STATE <= "101";
END IF;
WHEN "010" =>
IF (ENABLE_REG(2) = '1') THEN
DATA_ADC_OUT(7 downto 5) <= "000";
DATA_ADC_OUT(4) <= RNG_REG(2);
DATA_ADC_OUT(3) <= BIP_REG(2);
DATA_ADC_OUT(2 downto 0) <=

nCS_ADC <= '0';
nWR_ADC <= '0';
nRD_ADC <= '1';
CONVERSION_STATE <= "001";
ELSE
CONVERSION_STATE <= "101";
END IF;
WHEN "011" =>
IF (ENABLE_REG(3) = '1') THEN
DATA_ADC_OUT(7 downto 5) <= "000";
DATA_ADC_OUT(4) <= RNG_REG(3);
DATA_ADC_OUT(3) <= BIP_REG(3);
DATA_ADC_OUT(2 downto 0) <=

nCS_ADC <= '0';
nWR_ADC <= '0';
nRD_ADC <= '1';
CONVERSION_STATE <= "001";
ELSE
CONVERSION_STATE <= "101";
END IF;
WHEN "100" =>
IF (ENABLE_REG(4) = '1') THEN
DATA_ADC_OUT(7 downto 5) <= "000";
DATA_ADC_OUT(4) <= RNG_REG(4);
DATA_ADC_OUT(3) <= BIP_REG(4);
DATA_ADC_OUT(2 downto 0) <=

nCS_ADC <= '0';
nWR_ADC <= '0';
nRD_ADC <= '1';
CONVERSION_STATE <= "001";
ELSE
CONVERSION_STATE <= "101";
```



Center for Prototyping Services

Mechatronics Laboratory



```

                                END IF;
                                WHEN "101" =>
                                    IF (ENABLE_REG(5) = '1') THEN
                                        DATA_ADC_OUT(7 downto 5) <= "000";
                                        DATA_ADC_OUT(4) <= RNG_REG(5);
                                        DATA_ADC_OUT(3) <= BIP_REG(5);
                                        DATA_ADC_OUT(2 downto 0) <=

CONVERTER_CHANNEL;

                                        nCS_ADC <= '0';
                                        nWR_ADC <= '0';
                                        nRD_ADC <= '1';
                                        CONVERSION_STATE <= "001";

                                    ELSE
                                        CONVERSION_STATE <= "101";
                                    END IF;

                                WHEN "110" =>
                                    IF (ENABLE_REG(6) = '1') THEN
                                        DATA_ADC_OUT(7 downto 5) <= "000";
                                        DATA_ADC_OUT(4) <= RNG_REG(6);
                                        DATA_ADC_OUT(3) <= BIP_REG(6);
                                        DATA_ADC_OUT(2 downto 0) <=

CONVERTER_CHANNEL;

                                        nCS_ADC <= '0';
                                        nWR_ADC <= '0';
                                        nRD_ADC <= '1';
                                        CONVERSION_STATE <= "001";

                                    ELSE
                                        CONVERSION_STATE <= "101";
                                    END IF;
                                WHEN "111" =>
                                    IF (ENABLE_REG(7) = '1') THEN
                                        DATA_ADC_OUT(7 downto 5) <= "000";
                                        DATA_ADC_OUT(4) <= RNG_REG(7);
                                        DATA_ADC_OUT(3) <= BIP_REG(7);
                                        DATA_ADC_OUT(2 downto 0) <=

CONVERTER_CHANNEL;

                                        nCS_ADC <= '0';
                                        nWR_ADC <= '0';
                                        nRD_ADC <= '1';
                                        CONVERSION_STATE <= "001";

                                    ELSE
                                        CONVERSION_STATE <= "101";
                                    END IF;
                                WHEN OTHERS =>
                                    END CASE;
                                ELSIF (CONVERSION_STATE = "001") THEN -- End write control byte into
the ADC converter

                                    nCS_ADC <= '1';
                                    nWR_ADC <= '1';
                                    nRD_ADC <= '1';
                                    HBEN_ADC <= '1';
                                    CONVERSION_STATE <= "010";
                                ELSIF (CONVERSION_STATE = "010") THEN -- Wait for an interrupt from
the ADC converter

                                    IF (nINT_ADC = '0') THEN
                                        nCS_ADC <= '0';
                                        nWR_ADC <= '1';
                                        nRD_ADC <= '0';
                                        HBEN_ADC <= '0';
                                        CONVERSION_STATE <= "011";
                                    END IF;
                                ELSIF (CONVERSION_STATE = "011") THEN -- Read the LSB part of
converter response

                                CASE CONVERTER_CHANNEL IS
                                    WHEN "000" =>
```



Center for Prototyping Services

Mechatronics Laboratory



```
        ADC_REG(7 downto 0) <= DATA_ADC_IN;
    WHEN "001" =>
        ADC_REG(19 downto 12) <= DATA_ADC_IN;
    WHEN "010" =>
        ADC_REG(31 downto 24) <= DATA_ADC_IN;
    WHEN "011" =>
        ADC_REG(43 downto 36) <= DATA_ADC_IN;
    WHEN "100" =>
        ADC_REG(55 downto 48) <= DATA_ADC_IN;
    WHEN "101" =>
        ADC_REG(67 downto 60) <= DATA_ADC_IN;
    WHEN "110" =>
        ADC_REG(79 downto 72) <= DATA_ADC_IN;
    WHEN "111" =>
        ADC_REG(91 downto 84) <= DATA_ADC_IN;
    WHEN OTHERS =>
END CASE;
nCS_ADC <= '0';
nWR_ADC <= '1';
nRD_ADC <= '0';
HBEN_ADC <= '1';
CONVERSION_STATE <= "100";
ELSIF (CONVERSION_STATE = "100") THEN -- Read the MSB part of
converter response
    CASE CONVERTER_CHANNEL IS
        WHEN "000" =>
            ADC_REG(11 downto 8) <= DATA_ADC_IN(3 downto
0);
        WHEN "001" =>
            ADC_REG(23 downto 20) <= DATA_ADC_IN(3 downto
0);
        WHEN "010" =>
            ADC_REG(35 downto 32) <= DATA_ADC_IN(3 downto
0);
        WHEN "011" =>
            ADC_REG(47 downto 44) <= DATA_ADC_IN(3 downto
0);
        WHEN "100" =>
            ADC_REG(59 downto 56) <= DATA_ADC_IN(3 downto
0);
        WHEN "101" =>
            ADC_REG(71 downto 68) <= DATA_ADC_IN(3 downto
0);
        WHEN "110" =>
            ADC_REG(83 downto 80) <= DATA_ADC_IN(3 downto
0);
        WHEN "111" =>
            ADC_REG(95 downto 92) <= DATA_ADC_IN(3 downto
0);
        WHEN OTHERS =>
    END CASE;
    nCS_ADC <= '1';
    nWR_ADC <= '1';
    nRD_ADC <= '1';
    HBEN_ADC <= '1';
    CONVERSION_STATE <= "101";
    ELSIF (CONVERSION_STATE = "101") THEN -- Select the next ADC channel
or indicate EOC
        nCS_ADC <= '1';
        nWR_ADC <= '1';
        nRD_ADC <= '1';
        HBEN_ADC <= '1';
        CASE CONVERTER_CHANNEL IS
            WHEN "000" =>
                CONVERTER_CHANNEL <= "001";
                CONVERSION_STATE <= "000";
                END_OF_CONVERSION <= '0';
```



Center for Prototyping Services

Mechatronics Laboratory



```

        WHEN "001" =>
            CONVERTER_CHANNEL <= "010";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '0';
        WHEN "010" =>
            CONVERTER_CHANNEL <= "011";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '0';
        WHEN "011" =>
            CONVERTER_CHANNEL <= "100";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '0';
        WHEN "100" =>
            CONVERTER_CHANNEL <= "101";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '0';
        WHEN "101" =>
            CONVERTER_CHANNEL <= "110";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '0';
        WHEN "110" =>
            CONVERTER_CHANNEL <= "111";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '0';
        WHEN "111" =>
            CONVERTER_CHANNEL <= "000";
            CONVERSION_STATE <= "000";
            END_OF_CONVERSION <= '1';
            DELAY_END_OF_CONVERSION <= '1';
        WHEN OTHERS =>
            END CASE;
    END IF;
ELSE
    -- Initial state in which FSM is waiting for the start of conversion
    signal.
    -- Normally DELAY_END_OF_CONVERSION is '0' except when
    -- (CONVERSION_STATE="101" and
    CONVERTER_CHANNEL="111")
    -- otherwise a pulse is generated on END_OF_CONVERSION signal
    nCS_ADC <= '1';
    nWR_ADC <= '1';
    nRD_ADC <= '1';
    HBEN_ADC <= '1';
    CONVERSION_STATE <= "000";
    CONVERTER_CHANNEL <= "000";
    CASE DELAY_END_OF_CONVERSION IS
        WHEN '0' =>
            END_OF_CONVERSION <= '0';
        WHEN '1' =>
            DELAY_END_OF_CONVERSION <= '0';
            END_OF_CONVERSION <= '1';
        WHEN OTHERS =>
            END CASE;
    END IF;
END IF;
END PROCESS;

-- End of Architecture Processes Description
-----

END a;

-- End of Architecture Section
-----
```



Center for Prototyping Services

Mechatronics Laboratory



Document Author(s): Eduardo Miranda, Marcello Chiaberge

Document Title: **I/O Board Documentation**

File name: W:\OpenDSP_GeneralPurposeIO.doc

Created: 06-07-98 18:55

Last editing: 10-07-98 17:16

Page 25 of 25