8 The Ultragizmo Lab Board

This chapter provides detailed information about the various components on the University of Toronto Ultragizmo board.

The board itself is a printed circuit board, measures 12.5 x 7.0 inches, and runs at 16.67 MHz. Figure 42 shows where individual components are physically located on the board while Figure 2 (on page 2) is a functional block diagram of the board showing how the components are logically connected to each other.

The central processing unit on the board is the Motorola MC68306 16-bit integrated processor. This integrated processor contains an M68000 processor core plus several peripheral devices including a serial I/O controller and an interrupt controller. The M68000 has an asynchronous bus structure with a 16-bit data bus and a 32-bit address bus. Only the lower 24 bits of the address bus are connected to the external I/O of the MC68306 Chip. It has a powerful instruction set and versatile addressing modes. Chapter 6 describes how to write, assemble, and execute a program for the M68000, while the course textbook gives details about the instructions and addressing modes. A PC-based simulator for the M68000 is available in the directory *ugsparc:/cad2/ultragizmo/MLabs/68000sim*.

The system's main memory consists of 10 Mbytes of dynamic RAM (DRAM). The monitor program is stored in 2 Mbytes of flash ROM. The monitor program is a stripped-down operating system for the Ultragizmo board and provides commands for helping the programmer to debug incorrectly functioning programs. Chapter 7 summarizes the monitor commands available while Section 2.2 gives a tutorial and more details for each command.

All M68000 peripherals are memory-mapped. In other words, there are no special instructions for communicating between the M68000 and the various peripheral chips that control, for example, the serial and parallel ports. Instead, each peripheral chip is assigned a range of addresses and the M68000 communicates with that peripheral by reading and writing to the addresses assigned to that peripheral's registers.

The Ultragizmo board has two RS-232-C serial ports. Both are connected to a PC workstation. One serial port (CONSOLE) provides keyboard and screen I/O services for the board and another port (REMOTE) is used to download programs from the PC to the board. The serial ports are controlled by the Dual Asynchronous Receiver/Transmitter unit (DUART) of the MC68306 chip, which in turn is controlled by an M68000 program. Section 8.1 provides details on how to program the DUART to use the serial ports.

Interrupt controller of the MC68306 manages interrupts to the M68000 from the board's various communications devices. The M68000 has seven interrupt priority levels, which can be vectored or autovectored depending on the configuration of the interrupt controller. Section 8.2 describes the interrupt structure of the board as well as provides details on how to program the interrupt controller.

There are a total of 16 I/O lines (plus 5 control lines) available at a parallel interface/timer (PIT) to provide for parallel data transfer. The programmer can use the PIT to interface the M68000 with external hardware, such as a mouse or motorized LEGO kits. The PIT is controlled by the CFPGA. Section 8.3 provides details on how to access the PIT while Section 9.2, Section 9.3, and Chapter 10 describe how to use the PIT to control a mouse, a hex keypad or a LEGO board, respectively. A second PIT can be configured using the SFPGA. A procedure for

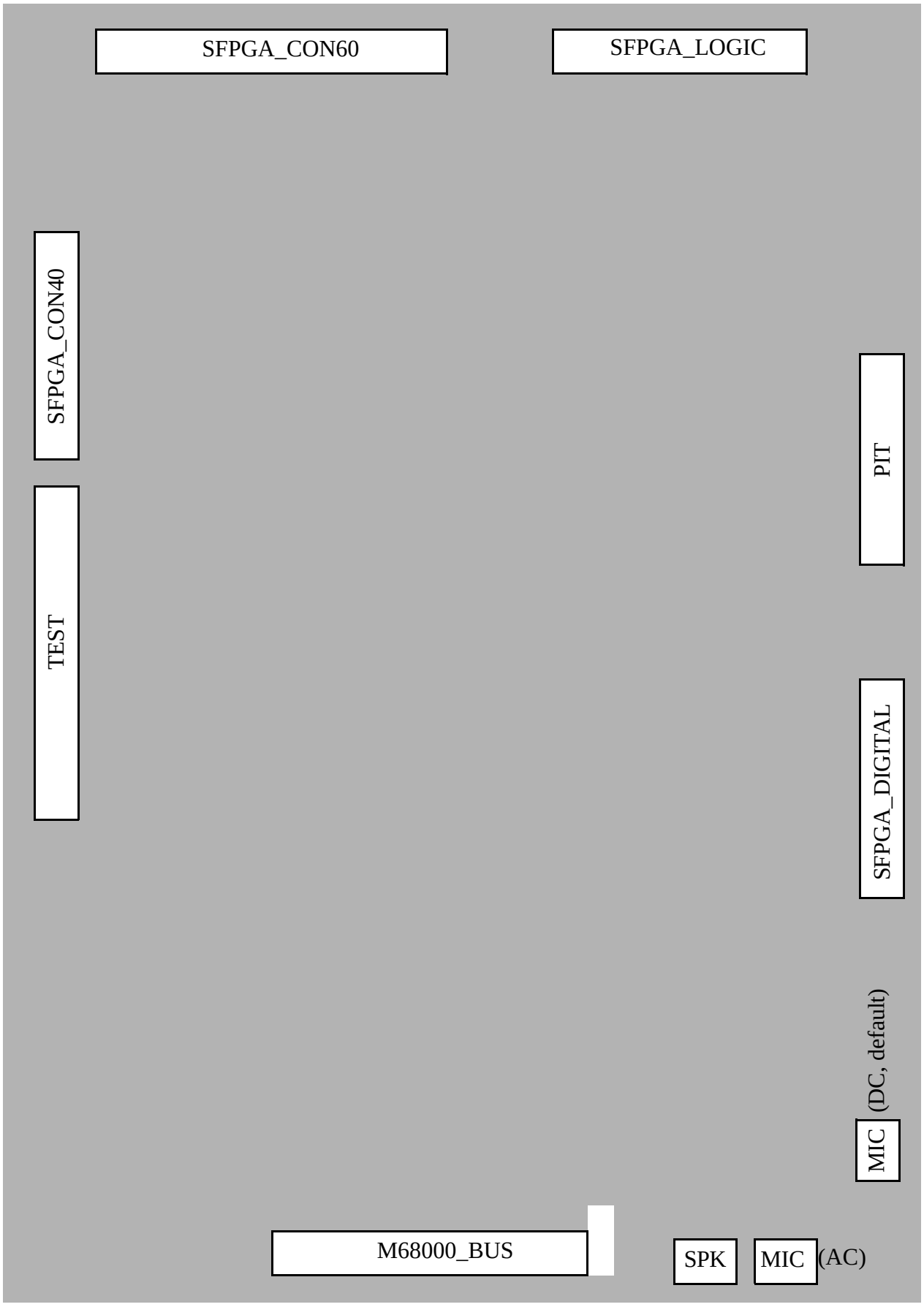

Figure 42 - *Placement of Components on Ultragizmo Board*

doing this is given in Section 8.3.3.

There is a completely buffered version of the main CPU bus available at a cable connector (**M68000_BUS**) so the user may add any additional peripheral controllers (or memory) needed for a given application. Section 8.4 provides the details on how to do this.

The SFPGA is an Altera 10K70 FPGA which can be used for prototyping purposes. Pins of the SFPGA are connected directly to the M68000 address and data buses. The SFPGA is also connected to and can be used to control a COder-DECoder (CODEC) for audio I/O, 16 Light Emitting Diodes (LED), four HEX displays (HEX1, HEX2) and 1Mbyte of SRAM modules. A Cypress ICD2053B programmable clock generator is used to provide clock signals for the SFPGA. External signals also can be connected to the SFPGA through four I/O connectors (**SFPGA_CON40**, **SFPGA_CON60**, **SFPGA_DIGITAL**, **SFPGA_LOGIC**). Section 8.5 describes the SFPGA and gives the pin assignments for the various connectors on the board. Section 8.6 describes the operation of the CODEC. Section 8.7 describes the operation of the programmable clock.

Finally, a word about accessing the 32-bit memory space of the M68000. The Ultragizmo board executes a program in supervisor mode. Hence, an M68000 program can use any of the M68000 instructions and access the entire 32-bit address space. This 4-Gbyte address space is divided among the various components on the board. Table 4 shows how the 32-bit address space is allocated. The user RAM (memory used to store your programs) starts at 0x8001. The text and data of a program can stretch from there to the top of the available physical memory, which is $009fffff for 10Mbyte of RAM.

Memory Addresses	Board Component
0x00000000 - 0x000003ff	Exception Vectors
0x00000400 - 0x00004fff	Monitor Variables
0x00005000 - 0x00008000	Stack Pointer location on reset
0x00008001 - 0x009fffff	User usable RAM
0x00a00000 - 0x00bfffff	Off Board Memory
0x00c00000 - 0x00cfffff	I/O Peripherals
0x00d00000 - 0x00dfffff	Flash ROM Programming
0x00e00000 - 0x00ffffff	Flash ROM
0x01000000 - 0xffffefff	Not used
0xfffff000 - 0xfffff7df	MC68306 Internal Registers
0xfffff7e0 - 0xfffff7ff	MC68306 DUART
0xfffff800 - 0xfffffff7	MC68306 Internal Registers
0xfffffff8 - 0xfffffffb	MC68306 Interrupt Controller
0xfffffffc - 0xffffffff	MC68306 Internal Registers

Table 4 - *Memory Map of the Ultragizmo Board*

8.1 DUART and Serial I/O

Besides the M68000 processor core, the MC68306 chip contains several subcomponents that it uses to communicate with the outside world. One of those subcomponents is the Dual Asynchronous Receiver/Transmitter unit, commonly referred to as the DUART. The DUART connects the M68000 to two serial RS-232-C ports (RS-232 for short). The RS-232 ports, in turn, are connected to the PC workstation, which is used by us to communicate with the M68000. Communicating using an RS-232 port is called **serial I/O**. This is because communication is done 1 bit at a time and each port can both receive (input) and send (output) data from and to the outside world. However, the M68000 does not receive data 1 bit at a time from an RS-232 port; that is the job of the DUART. Instead, an RS-232 port sends a serial (1-bit) data stream from the outside world to the DUART which then converts this into a parallel, byte-wide (8-bits) datum that is sent on the data bus to the M68000. The same process is used in reverse for sending a serial data stream to the outside world.

The DUART has two ports, named A and B. Each port controls one of the RS-232 ports. Figure 2 on page 2 shows the connections between the DUART and the RS-232 ports. Port A (REMOTE) on the DUART controls the RS-232 port that is used for downloading M68000 programs from the PC while Port B (CONSOLE) controls the RS-232 port that provides keyboard and screen I/O for the board.

All communication between the DUART and the M68000 uses byte transfers. This is because the DUART is an 8-bit bus device that is attached to the low-order byte of the data bus. Communication with the DUART is done by writing or reading any of the registers in the DUART. The Ultragizmo board uses memory-mapped I/O so accessing the DUART's register is done by accessing the appropriate memory location. Table 5 lists the DUART registers and gives the addresses of the registers for each DUART port. Note that a "register" is sometimes used differently depending on whether it is being read or written. For example, the register at address 0xfffff7e7 is a receiving buffer when it is being read and a transmitting buffer when it is being written. Internally the DUART has separate receive and transmit buffer registers; it determines which is being accessed by looking at the read-write (R/$\overline{W}$) line coming from the M68000 in addition to the address lines.

8.1.1 Communicating with the Terminal (Using Port B on DUART)

Port B on the DUART is connected to the PC for keyboard and screen I/O and is the port that is used most often by programmers. Hence, we will explain how to use the DUART to communicate with the PC through port B; the same procedure will allow you to communicate through the other RS-232 ports. The monitor program automatically initializes Port B on the DUART to the settings necessary to communicate with the PC, so no initialization code is needed. Initialization code will be needed to use the other ports; a description of the required code is given later in Section 8.1.3.

Characters written to Port B of the DUART by the M68000 appear on the **CONPORT** window, while reading from Port B lets the M68000 determine which keys were pressed. Writing a character on the screen is done by writing to the transmitter buffer B (TBB) at address 0xfffff7f7. Reading from this same address will read the receiving buffer for Port B (RBB) and produce the most recently pressed key on the PC keyboard while the **CONPORT** window is in focus. The **CONPORT** window understands characters encoded in ASCII, so writing the byte

DUART	READ	(name)	WRITE	(name)
0xfffff7e1	Mode Register A	(MR1A, MR2A)	Mode Register A	(MR1A, MR2A)
0xfffff7e3	Status Register A	(SRA)	Clock-Select Register A	(CSRA)
0xfffff7e5	*Reserved*		Command Register A	(CRA)
0xfffff7e7	Receiver Buffer A	(RBA)	Transmitter Buffer A	(TBA)
0xfffff7e9	Input Port Change Register	(IPCR)	Auxiliary Control Register	(ACR)
0xfffff7eb	Interrupt Status Register	(ISR)	Interrupt Mask Register	(IMR)
0xfffff7ed	Current MSB of Counter	(CUR)	Counter/Timer Upper Register	(CTUR)
0xfffff7ef	Current LSB of Counter	(CLR)	Counter/Timer Lower Register	(CTLR)
0xfffff7f1	Mode Register B	(MR1B, MR2B)	Mode Register B	(MR1B, MR2B)
0xfffff7f3	Status Register B	(SRB)	Clock-Select Register B	(CSRB)
0xfffff7f5	*Reserved*		Command Register B	(CRB)
0xfffff7f7	Receiver Buffer B	(RBB)	Transmitter Buffer B	(TBB)
0xfffff7f9	Interrupt Vector Register	(IVR)	Interrupt Vector Register	(IVR)
0xfffff7fb	Input Port (unlatched)		Output Port Config. Register	(OPCR)
0xfffff7fd	Start Counter Command	(STC)	Set Output Port Register	(SOPR)
0xfffff7ff	Stop Counter Command	(SPC)	Reset Output Port Register	(ROPR)

Table 5 - *Register Addresses for DUART*

0x41 to 0xfffff7f7 would print the character ‘A’ on the screen while reading the byte 0x37 means that the ‘7’ key was pressed. Table D.2 of **Computer Organization** by Hamacher, Vranesic and Zaky, 1996, 4th edition lists all the ASCII character encodings.

The M68000 can process characters at a much faster rate than the DUART can. For example, if the M68000 were to continually send characters to the screen, not all the characters would be displayed because the screen would display one character in the time it takes the M68000 to send several. These subsequent characters would be overwritten in the transmit buffer and thus be lost before the first character had finished being displayed.

One way to avoid this is to have the M68000 check the DUART’s Status Register for Port B (SRB at address 0xfffff7f3) to see when the screen has finished displaying a character and is ready to display another character. Table 6 shows the format of the Status Register (see Chapter 6 in the **MC68306 User’s Manual** to find out what the other bits are used for). Bit 2 of the status register is set to 0 when the transmitter buffer (TBB) is full; data written to the transmitter buffer by the processor has not yet been sent out to the serial port. This bit is set to 1 when the DUART has

	7	6	5	4	3	2	1	0	
Status Register (SR)						TxRdy		RxRdy	=0 no =1 yes
Interrupt Mask Register (IMR)			RxRdyB	TxRdyB			RxRdyA	TxRdyA	=0 disabled =1 enabled
Interrupt Status Register (ISR)			RxRdyB	TxRdyB			RxRdyA	TxRdyA	=0 didn't occur =1 occurred

Table 6 - *Status and Interrupt Registers on the DUART*

room in the transmitter buffer to accept another byte of data from the processor.

A similar problem occurs for reading characters from the keyboard; bit 0 of SRB is used to solve the problem. This bit is a 1 whenever there is data in the receiver buffer (RBB) that has not yet been read by the processor. When the receiver buffer is read, this bit is cleared automatically by the DUART.

Following is a simple example showing how to use the status register and transmitter/receiver register to echo typed characters onto the terminal:

```
SRB equ $fffff7f3
RBB equ $fffff7f7
TBB equ $fffff7f7

        org $20000

TSTRX   btst.b #0,SRB           Character received?
        beq TSTRX
        move.b RBB,d0
TSTTX   btst.b #2,SRB           Transmitter ready?
        beq TSTTX
        move.b d0,TBB
        bra TSTRX
```

The above code is available in */cad2/ultragizmo/MLabs/duart_poll.s* on the *ugsparc* system. Using the status register in this way to control the flow of bytes between the M68000 and the terminal is called **polled I/O** because the processor uses a tight loop to continually poll the DUART until the DUART is ready.

In addition to writing characters on the screen, it is also possible to clear the screen and move the screen cursor to a desired position to do "formatted printing." This is done by writing special escape sequences to the terminal. Some of these sequences are shown in Table 7. Each character in a sequence is to be written using its ASCII encoding. The spaces in the character sequences are for readability only; do not send them. For example, to clear the screen, write the four bytes 0x1b, 0x5b, 0x32, and 0x4a to register TBA using polled I/O. To move the cursor to a particular spot, say the 3rd row and 10th column, use the second control sequence. Remember to write the ASCII encoding of these numbers to the screen and not their binary representation! Refer to a VT-52 or

Esc [H	Move the cursor to the home position (line 1, column 1).
Esc [Pl ; Pc H	Move the cursor to line Pl and column Pc. Pl and Pc are ASCII decimal strings.
Esc [2 K	Erase the current line.
Esc [2 J	Erase the entire display

Table 7 - *Terminal Control Sequences*

VT-100 manual for more information on the control characters.

Reading and writing Port A of the DUART can be done in a manner similar to the above; initialization of the DUART is explained in Section 8.1.3 on page 98.

8.1.2 DUART Autovectored Interrupts at Level 4

Obviously, using polled I/O can be a waste of a processor's compute power. Rather than having the M68000 processor constantly checking to see if the I/O device is ready, the I/O device could interrupt the M68000 when it is ready and the M68000 could perform some other task while waiting for the I/O device's interrupt signal. This type of I/O is called **interrupt-driven**. For example, the DUART can generate an interrupt to inform the M68000 that it is ready to transmit another character or that it has received a new character. The DUART can generate four different interrupts, two for each port:

Interrupt	**Meaning**
RxRdyA	new character received on channel A
TxRdyA	ready to transmit character on channel A
RxRdyB	new character received on channel B
TxRdyB	ready to transmit character on channel B

Section 8.2 provides a general explanation on setting up and processing interrupts. This section provides some specific details for enabling, disabling, and clearing interrupts from the DUART on interrupt level 4.

By default, an autovectored interrupt from the DUART is signaled at interrupt level 4. Interrupts from the DUART are enabled and disabled using the Interrupt Mask Register (IMR). To enable interrupts, appropriate bits in the IMR must be set to 1. To disable interrupts, those bits must be set to 0. Four bits in the IMR enable and disable the four different interrupts a DUART can generate. Bits 0 and 1 of the IMR control the generation of the TxRdyA and RxRdyA interrupts, respectively, while bits 4 and 5 control the TxRdyB and RxRdyB interrupts, respectively (see Table 6).

In addition to enabling interrupts by setting the DUART's interrupt mask correctly, the Interrupt Vector Register (IVR) must be filled with the correct interrupt vector number. For the default autovectored interrupt at level 4, this is done by setting IVR to 28, the interrupt vector

number for level 4 autovectored interrupts.

Once a DUART interrupt has been signaled, an interrupt service routine executing on the M68000 can determine what type of interrupt has occurred by examining the DUART's interrupt status register (ISR). Four bits in the ISR corresponding to those in the IMR indicate which particular interrupt occurred (see Table 6).

Finally, to clear a DUART interrupt, the appropriate bit in the DUART's ISR should be set to 0. This is automatically done when the corresponding register is read or written. For example, when a **TxRdyA** interrupt is signaled and the interrupt service routine writes the TBA register to send a character to the ready terminal, bit 0 of the ISR is cleared automatically.

The following is code for configuring Port B of DUART to generate an interrupt when the receiver is ready:

```
IMR     equ         $fffff7eb ;interrupt mask register
IVR     equ         $fffff7f9 ;interrupt vector register
        org $20000
        move.l #inter,$70     ;initialize interrupt vector table
        move.b #28,IVR        ;initialize IVR
        move.b #%00100000,IMR ;initialize IMR for interrupt on Receiver
Ready
        move.w #$2300,sr
        . . .
inter   . . .                 ;interrupt service routine
```

Please refer to Section 8.2.4 for different DUART interrupt level configurations.

8.1.3 DUART Initialization

The DUART used on the Ultragizmo board is very flexible. The number of start and stop bits, baud rate and various other communication options can be varied easily. This is done by writing appropriate values to the DUART's various mode and control registers. The **MC68306 User's Manual** provides a complete description of the DUART registers in section 6.

One unusual feature of the mode registers is that two registers, MR1x and MR2x, appear at the same address (we use x to mean A or B when the discussion applies to both ports). The first write to this address after a reset goes to MR1x; subsequent writes go to MR2x until the pointer is reset (by writing to yet another register!).

The following is code for configuring Port B of the DUART:

```
*
* INIT. DUART Port B
* 8 data bits, 1 stop bit, 9600 baud, no parity
* op4 and op6 interrupt outputs txrdya and rxrdya
*
MR1B equ $fffff7f1
MR2B equ $fffff7f1
CSRB equ $fffff7f3
CRB equ $fffff7f5
ACR equ $fffff7e9
```

```
org $20000

move.b #$1a,CRB  ;Disable Rx, Tx, and reset mode pointer
move.b #$30,CRB  ;Reset transmitter(tx)
move.b #$20,CRB  ;Reset receiver(rx)
move.b #$13,MR1B ;8 bits, no parity
move.b #$07,MR2B ;1 stop bit; disable hardware flow control
move.b #$bb,CSRB ;9600 baud tx and rx
move.b #$60,ACR  ;Baud rate set select
move.b #$05,CRB  ;Enable rx, tx
```

The above code is available in */cad2/ultragizmo/MLabs/duart_init.s* on the *ugsparc* system. To initialize the mode and control registers for the serial ports, use the above as a guide and refer to the **MC68306 User's Manual** for a complete register description.

8.2 Interrupts on the Ultragizmo Board

Efficient communication between the processor and peripherals can be implemented using interrupts. This section discusses the interrupt system architecture of the Ultragizmo board.

For external interrupts (interrupts generated by devices external to the MC68306 processor), the MC68306 processor implements the entire 7 interrupt levels of the M68000 instruction set, providing both vectored and autovectored interrupts through an on-chip interrupt controller. Internal interrupts (interrupts generated by on-chip peripherals like the DUART and the 16-bit DUART Timer) are not controlled by the interrupt controller. Instead, dedicated system registers are used to control these interrupts. Both external and internal interrupts are discussed in this section. Motorola documentation often refers to *interrupts* as *exceptions*. We use the term *interrupt* exclusively to avoid confusion but the two terms can be considered interchangeable for the purposes of this discussion.

Section 8.2.1 discusses the basic concepts regarding the M68000 interrupts and the interrupt controller for the MC68306 processor, without considering the limitations imposed by the Ultragizmo architecture. The Ultragizmo board and its operating system allow user programs and external devices to access a subset of M68000 interrupts. The Ultragizmo interrupt architecture is discussed in Section 8.2.2. Section 8.2.3 summarizes the basic procedure for initializing external interrupts on the Ultragizmo board. It also provides several programming examples.

Internal interrupts are discussed in Section 8.2.4 and Section 8.2.5. Section 8.1.2 specifically discusses generating level 4 autovectored interrupts using the DUART, whereas Section 8.2.4 discusses generating DUART interrupts at other interrupt levels. The 16-bit DUART Timer on the MC68306 can be used to interrupt the processor at fixed time intervals. It is discussed in Section 8.2.5.

8.2.1 M68000 Interrupts and the Interrupt Controller of MC68306

The M68000 instruction set has seven interrupt priority levels, numbered from 1 to 7 with priority 7 being the highest. The interrupt signals are managed by the interrupt control unit on the MC68306 chip. An I/O device, such as a DUART or PIT, can interrupt the M68000 at a particular priority level and cause the M68000 to execute an **interrupt service routine** that handles the event causing the interrupt. To begin executing the interrupt service routine, the M68000 needs to know the starting address of the routine.

The starting address of an interrupt service routine is called an **interrupt vector**. Interrupt vectors are stored in the Interrupt Vector Table which is placed in a fixed location in memory. For the M68000, the Interrupt Vector Table begins at address 0 and stores 256 addresses. Table 8 shows the part of the M68000 Interrupt Vector Table that is of interest to us.

An entry in the Interrupt Vector Table corresponds to a particular type of interrupt. For example, entry 25 at address 100 (=0x64) stores the address of the interrupt service routine that executes in response to a priority 1 autovectored interrupt. Because all addresses are 4 bytes long, the M68000 uses the entry number to identify the location of the address of an interrupt service routine in the Interrupt Vector Table. This identifier is called a **vector number** or **vector** for short. When you define an interrupt service routine for a particular interrupt event, you must store the address of the service routine (i.e., its interrupt vector) at the appropriate vector in the Interrupt Vector Table. Note the distinction between a *vector* and an *interrupt vector*!

Vector Number	Address	Assignment
0-24	0-96 (=0x00-0x60)	Reserved
25	100 (=0x64)	priority 1 autovectored interrupt
26	104 (=0x68)	priority 2 autovectored interrupt
27	108 (=0x6c)	priority 3 autovectored interrupt
28	112 (=0x70)	priority 4 autovectored interrupt
29	116 (=0x74)	priority 5 autovectored interrupt
30	120 (=0x78)	priority 6 autovectored interrupt
31	124 (=0x82)	priority 7 autovectored interrupt
31-63	128-252 (=0x86-0x96)	Reserved
64–255	256–1020 (=0x100–0x3FC)	vectored interrupts

Table 8 - *Overview of the M68000 Interrupt Vector Table*

There are two different methods for specifying the vector number of an interrupt: **autovectored** and **vectored** interrupts. Autovectored interrupts use the priority level to specify the vector number. The interrupting I/O device does not supply a vector and instead the M68000 generates a vector depending on the priority of the interrupt. For example, as Table 8 shows, the M68000 generates vector number 25 for priority 1 autovectored interrupts and vector number 31 for priority 7 autovectored interrupts. In general the vector number for an autovectored interrupt at priority *p* is 24+*p*.

In contrast, devices generating vectored interrupts must provide the vector explicitly. The interrupting device must generate the 8-bit vector number and send it to the M68000 over the data bus when requested by the processor in a vectored interrupt cycle. The M68000 defines the first 64 vectors for its own purposes, including the autovectored interrupts, and leaves the remaining 192 vectors for the system designer to allocate. When using vectored interrupts, the generated vector must be one of these 192 vectors.

Autovectored interrupts allow I/O devices to interrupt the M68000 with almost no extra hardware thus allowing hardware designers to easily add an I/O device with interrupting capabilities. Vectored interrupts, on the other hand, allow more than 7 events to cause an interrupt and to still be serviced efficiently.

Each MC68306 processor contains an on-chip interrupt control unit. This controller decodes the standard M68000 interrupt signals into seven I/O signal pairs, one for each interrupt level. The two signals in each pair are the interrupt request signal ($\overline{\text{IRQ}}$) and the interrupt acknowledge signal ($\overline{\text{IACK}}$). The $\overline{\text{IRQ}}$s are inputs to the processor, and the $\overline{\text{IACK}}$s are outputs from the processor. An external device can request an interrupt by lowering an $\overline{\text{IRQ}}$ line. The $\overline{\text{IACK}}$ signals are used

Interrupt Control Register (ICR) Address: FFFFFFFA/B

FFFFFFFA	IENT	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1
Reset :	0	0	0	0	0	0	0	0
FFFFFFFB	-----	AVEC7	AVEC6	AVEC5	AVEC4	AVEC3	AVEC2	AVEC1
Reset :	1	1	1	1	1	1	1	1

IENT --- Timer Interrupt Enable
This bit enables the DUART timer interrupt
0 - Interrupts disabled
1 = Interrupts enabled

IEN7-1 --- Interrupt Enable 7 through 1
These bits enable interrupt 7,6,5,4,3,2,1
0 = Interrupts disabled
1 = Interrupts enabled

AVEC7-1 --- Autovector Enable 7 through 1
These bits enable autovectoring for interrupts 7,6,5,4,3,2,1
0 = No autovector
1 = Autovector

Figure 43 - *Interrupt Control Register (ICR)*

for handshaking during the vectored interrupts. For a complete description of these signals, please refer to the **MC68306 User's Manual**.

Whether a priority level uses vectored or autovectored interrupts is also determined by the interrupt controller. The interrupt controller contains a 16-bit Interrupt Control Register (ICR), which can be configured by software. Seven bits of the ICR indicate if the corresponding interrupt levels are vectored or autovectored. Another 7 bits enable or disable their corresponding interrupt levels. Finally, one bit enables the timer interrupt, which is described in Section 8.2.5 on page 107. Figure 43 shows the configuration of the ICR in detail.

The $\overline{\text{IRQ}}$ and $\overline{\text{IACK}}$ signals of interrupt levels 2, 3, 5, and 6, are multiplexed with other signals at the MC68306 I/O pins. The configuration of these pins is controlled by two 16-bit MC68306 internal registers, Port Direction Register (PDIR) and Port Data Register (PDATA). The PDIR register is located at address $FFFFFFF2. The PDATA register is located at address $FFFFFFF0. The configuration of these two registers for each interrupt level is listed in Table 9. For a detailed description of the PDIR and PDATA registers, please refer to page 5-7 and 5-8 of the MC68306 User's Manual.

8.2.2 Interrupt Signal Assignments on the Ultragizmo Board

This section describes specifics and limitations of interrupts on the Ultragizmo board. On the Ultragizmo board, for level 3 and 5 interrupts, there are no $\overline{\text{IACK}}$ connections to the outside world, only $\overline{\text{IRQ}}$ connections; thus level 3 and 5 interrupts can only be used as autovectored interrupts by external devices. For interrupt levels 1, 2 and 6, both $\overline{\text{IRQ}}$ and $\overline{\text{IACK}}$ signals are connected to external connections. These interrupt levels can be used either as autovectored interrupts or vectored interrupts. Figure 44 shows the interrupt signal connections on the Ultragizmo board in detail. The same information is summarized in Table 10.

Interrupt Level and Type	PDIR ($FFFFFFF2)	PDATA ($FFFFFFF0)
Level 2 - autovector	$bit_4 = 0$	$bit_0 = 0$
Level 2 - vector	$bit_4 = 0$, $bit_0 = 1$	--------
Level 3 - autovector	$bit_5 = 0$	$bit_1 = 0$
Level 3 - vector	$bit_5 = 0$, $bit_1 = 1$	--------
Level 5 - autovector	$bit_6 = 0$	$bit_2 = 0$
Level 5 - vector	$bit_6 = 0$, $bit_2 = 1$	--------
Level 6 - autovector	$bit_7 = 0$	$bit_3 = 0$
Level 6 - vector	$bit_7 = 0$, $bit_3 = 1$	--------

Table 9 - *PDIR and PDATA Configurations*

Level	Type of Interrupt	CFPGA_ICR ($c20000)	Interrupt Event	$\overline{\text{IRQ}}$	$\overline{\text{IACK}}$
7	autovector only	-------	$\overline{\text{IRQ}}$ low	NMI / DUART Timer	NMI
6	vector / autovector	$06	$\overline{\text{IRQ}}$ low	pin AV20 ($\overline{\text{IRQ}}$) of SFPGA	pin D12 ($\overline{\text{IACK}}$) of SFPGA
5	autovector only	$05	$\overline{\text{IRQ}}$ high to low transition	pin 10 (H1), pin 12 (H2), pin 14 (H3), or pin 16 (H4) of PIT connector	-------
5	autovector only	$f5	$\overline{\text{IRQ}}$ low to high transition	pin 10(H1), pin 12 (H2), pin 14 (H3), or pin 16 (H4) of PIT connector	-------
4	vector / autovector	-------	$\overline{\text{IRQ}}$ low	DUART	DUART
3	autovector only	-------	$\overline{\text{IRQ}}$ low	pin 43 ($\overline{\text{AUTO3}}$) of M68000_BUS connector	-------
2	vector / autovector	-------	$\overline{\text{IRQ}}$ low	pin 44 ($\overline{\text{IRQ2}}$) of M68000_BUS connector	pin 42 ($\overline{\text{IACK2}}$) of M68000_BUS connector
1	vector / autovector	$01	$\overline{\text{IRQ}}$ low	pin 10 (H1) of PIT connector	pin 12 (H2) of PIT connector

Table 10 - *Interrupt Priorities on the Ultragizmo Board*

As shown in Figure 44, interrupt signals of level 2 and 3 are directly connected to the M68000 bus. These interrupts are always enabled. Interrupt signals of level 1, 5 and 6, however, are controlled by the CFPGA. The CFPGA contains an 8-bit configuration register (CFPGA_ICR) at address $c20000. Currently, the CFPGA can only service one interrupt level at a time. This interrupt level is determined by the contents of the configuration register as listed in Table 10. For

Figure 44 - *Overview of Interrupt Control on the Ultragizmo Board*

example, when the configuration register contains value $01, a 0 on pin 10 (H1) of PIT will cause the CFPGA to generate a level 1 interrupt request; level 5 and 6 interrupts are disabled.

All interrupts, except level 5 interrupts, are level sensitive and active low. Level 5 interrupts are edge triggered either by high-to-low or low-to-high transitions. Columns 3 and 5 of Table 10 show the CFPGA_ICR configuration for each of these two types.

For vectored interrupts at interrupt level 1 or 6, the CFPGA automatically handles the interrupt acknowledgment cycles. The interrupt vector number provided by the CFPGA is the value contained in the 8-bit CFPGA Interrupt Vector Register (CFPGA_IVR) at address location $c20002. This value is set by user programs.

8.2.3 Interrupt Initialization

To enable an interrupt on the Ultragizmo board, follow these seven steps:

1. If the interrupt is at level 1, 5, or 6, initialize the CFPGA_ICR register with proper values from Table 10.
2. If the interrupt is a vectored interrupt at level 1 or 6, initialize the CFPGA_IVR register with the corresponding interrupt vector number as discussed in Section 8.2.2.
3. Initialize the interrupt vector table as discussed in Section 8.2.1.
4. If the interrupt is at level 2, 3, 5, or 6, enable the $\overline{\text{IRQ}}$ and $\overline{\text{IACK}}$ lines by properly initializing the PDIR register as described in Table 9. Note: All bits of the PDIR register contain a default value of 0.
5. If the interrupt is not a level 4 interrupt, initialize the ICR register as described in Figure 43 on page 102. Note: The ICR register is configured for level 4 interrupt by default.

6. If the interrupt is autovectored at level 2, 3, 5, or 6, initialize the PDATA register as described in Table 9. Note: The reset value of the PDATA register is undefined.
7. Set the processor status register (SR) to a proper priority level.

We conclude this subsection by three interrupt programming examples. Each program continuously prints the character ‘N’ on the screen until it is interrupted by the corresponding interrupt signal. Once interrupted, a interrupt service routine prints the character ‘I’ for 100 times. Then the program returns to the main program and continuously prints the character ‘N’ again.

8.2.3.1 Vectored Interrupt at Level 1

```
SRB               equ $FFFFF7F3 ;duart status register
TBB               equ $FFFFF7F7 ;duart transmit register
ICR               equ $FFFFFFFA ;interrupt register - cpu
CFPGA_ICR         equ $00c20000 ;CFPGA interrupt configuration register
CFPGA_IVR         equ $00c20002 ;CFPGA interrupt vector register
N                 equ $4E
I                 equ $49

* initializing vectored interrupt at level 1
        org     $20000
        move.b  #$01,CFPGA_ICR       ;step 1 (see sec 8.2.3)
        move.b  #64,CFPGA_IVR        ;step 2 (see sec 8.2.3)
        move.l  #inter,$100          ;step 3 (see sec 8.2.3)
        move.w  #$41fe,ICR           ;step 5(see sec 8.2.3)
        move.w  #$2000,sr            ;step 6 (see sec 8.2.3)

* main program loop - continuously print ‘N’
loopt   btst.b  #2,SRB
        move.b  #N,TBB
        bra     loopt

* interrupt subroutine - print 100 ‘I’
        org     $30000
inter   move.b  #I,d0
        move.l  #100,d1
loopi   btst.b  #2,SRB
        beq     loopi
        move.b  d0,TBB
        dbra    d1,loopi
        rte
```

8.2.3.2 Vectored Interrupt at Level 2

```
SRB     equ $FFFFF7F3 ;duart status register
TBB     equ $FFFFF7F7 ;duart transmit register
ICR     equ $FFFFFFFA ;interrupt control register - cpu
PDIR    equ $FFFFFFF2 ;port direction register - cpu
PDATA   equ $FFFFFFF0 ;port data register - cpu
N       equ $4E
I       equ $49
```

```
* initializing vectored interrupt at level 2
        org     $20000
        move.l  #inter,$100  ;step 3 (see sec 8.2.3)
        move.w  #$1,PDIR     ;step 4 (see sec 8.2.3)
        move.w  #$42fd,ICR   ;step 5 (see sec 8.2.3)
        move.w  #$fffe,PDATA ;step 6 (see sec 8.2.3)
        move.w  #$2100,sr    ;step 7 (see sec 8.2.3)

* main program loop - continuously print 'N'
loopt   btst.b  #2,SRB
        move.b  #N,TBB
        bra     loopt

* interrupt subroutine - print 100 'I'
        org     $30000
inter   move.b  #I,d0
        move.l  #100,d1
loopi   btst.b  #2,SRB
        beq     loopi
        move.b  d0,TBB
        dbra    d1,loopi
        rte
```

8.2.3.3 Autovectored Interrupt at Level 5

```
SRB                 equ $FFFFF7F3 ;duart status register
TBB                 equ $FFFFF7F7 ;duart transmit register
ICR                 equ $FFFFFFFA ;interrupt control register - cpu
CFPGA_ICR           equ $00c20000 ;CFPGA interrupt configuration register
PDATA               equ $FFFFFFF0 ;port data register - cpu
N                   equ $4E
I                   equ $49

* initializing autovectored interrupt at level 5
* step 4 is skipped since we are using the default value of PDIR ($0000)
        org     $20000
        move.b  #$05,CFPGA_ICR   ;step 1 (see sec 8.2.3)
        move.l  #inter,$74       ;step 3 (see sec 8.2.3)
        move.w  #$50ff,ICR       ;step 5 (see sec 8.2.3)
        move.w  #$fffb,PDATA     ;step 6 (see sec 8.2.3)
        move.w  #$2200,sr        ;step 7 (see sec 8.2.3)
* main program loop - continuously print 'N'
loopt   btst.b  #2,SRB
        move.b  #N,TBB
        bra     loopt
* interrupt subroutine - print 100 'I'
        org     $30000
inter   move.b  #I,d0
        move.l  #100,d1
loopi   btst.b  #2,SRB
        beq     loopi
        move.b  d0,TBB
        dbra    d1,loopi
        rte
```

8.2.4 Generating Interrupts with the DUART

Section 8.1.2 discusses how to generate autovectored level 4 interrupts using the DUART. By default, interrupt level 4 is assigned to the DUART, but the DUART also can be configured to use other interrupt levels. This is controlled by the last three bits of the System Register (SREG) at address location $FFFFFFFE. Table 11 lists the configuration values and their corresponding interrupt levels. These three bits have the default value of 100, which sets the DUART interrupt level to level 4 by default.

$SREG_{2-0}$ Value	Interrupt Level
000	Reserved
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 11 - *DUART Interrupt Level Configuration*

To configure the DUART to generate an autovectored interrupt, the DUART's IVR (see Table 5) should contain the corresponding vector number for the given autovector interrupt level as shown in Table 8. For example, for the default level 4 interrupt, the IVR should be configured to be 28. For level 5 autovectored interrupt, the IVR should be configured to be 27.

The DUART also can be configured to generate vectored interrupts by setting the IVR value to be a value between 64 and 255, which is the vectored interrupt section of the interrupt vector table.

8.2.5 Generating Interrupts with the 16-Bit DUART Timer

The MC68306 contains a 16-bit timer, which is part of the DUART. The timer is described in detail in the **MC68306 User's Manual** on page 6-16 and page 6-17. This section gives an overview of the timer. The timer can be used to periodically generate interrupts, or output a periodical square wave on the MC68306 output pin, OP3, which is connected to pin 37 of the PIT (Section 8.3). The functionality and two operation modes are controlled by a set of registers listed in Table 12.

The Timer Vector Register (TVR) is an 8-bit register. The value of TVR determines the interrupt vector number of the timer, when a vectored timer interrupt is configured. Since the timer is part of the DUART, it can also be configured through the DUART, which is set to level 4 on the

Address	Register	Name	Width (bits)
$FFFFFFFF	timer vector register	TVR	8
$FFFFFFFA	interrupt control register	ICR	16
$FFFFF7FF	timer stop counter command register	SPC	8
$FFFFF7FD	timer start counter command register	STC	8
$FFFFF7F9	DUART interrupt vector register	IVR	8
$FFFFF7EF	timer counter upper preload register	CUR	8
$FFFFF7ED	timer counter lower preload register	CLR	8
$FFFFF7EB	DUART interrupt mask register	IMR	8
$FFFFF7E9	DUART auxiliary control register	ACR	8

Table 12 - *16-bit DUART Timer Configuration Registers*

ultragizmo board. Two programming examples are supplied at the end of this section to demonstrate these two distinct interrupt methods.

The ICR register is described in Section 8.2.1. The left most bit of the ICR register should be set to 1 if a TVR controlled timer interrupt is needed; the TVR register is used as the interrupt vector number. When the DUART autovectored interrupt is desired, this bit should be set to 0.

The fourth bit of the IMR register (Section 8.1.1) enables and disables the timer interrupts. A value of 1 enables the timer interrupt, and a value of 0 disables the timer interrupt. The period of the timer interrupts is controlled by the CUR and CLR registers. These two registers are treated as a single 16-bit register, with the CUR register as the upper byte and the CLR as the lower byte. The processor decrements the 16-bit register every DUART clock cycle, which can be configured to several distinct clock frequencies (see the **MC68306 User's Manual** for instructions), and an interrupt is generated when the value of the 16-bit register is reduced to zero. Once the interrupt is serviced, the 16-bit register is reset to its starting value and the entire cycle is repeated.

The timer is started when the user program reads the STC register. It can be stopped by reading the SPC register. The values of these two registers have no meaning. The ACR register controls additional timer configurations. Refer to page 6-30 of the **MC68306 User's Manual** for details.

The following two examples demonstrate the process of configuring the timer for vectored and autovectored interrupts. The vectored interrupt is shown in example 1. The timer periodically interrupts the processor at level 7. The autovectored interrupt is shown in example 2. The timer periodically interrupts the processor at level 4. The main loop of both programs repeatedly prints the character 'N'. The interrupt service routine prints 100 'I's on the screen.

8.2.5.1 Example 1 - Vectored Timer Interrupts

```
SRB     equ     $FFFFF7F3 ;duart status register
TBB     equ     $FFFFF7F7 ;duart transmit register
ICR     equ     $FFFFFFFA ;interrupt control register - cpu
TVR     equ     $FFFFFFFF ;timer vector register
IMR     equ     $FFFFF7EB ;duart interrupt mask register
ACR     equ     $FFFFF7E9 ;duart auxiliary control register
CUR     equ     $FFFFF7EF ;timer counter upper preload register
CLR     equ     $FFFFF7ED ;timer counter lower preload register
STC     equ     $FFFFF7FD ;timer start counter command register
SPC     equ     $FFFFF7FF ;timer stop counter command register
tmp     equ     $31000    ;temporary variable
N       equ     $4E
I       equ     $49

* Using the timer to generate a vectored interrupt
* at priority 7 (Non-Maskable)
        org     $20000
        move.l  #inter, $100  ;initialize the interrupt vector table
        move.b  #64,TVR       ;initialize the timer vector register
        move.w  #$c000,ICR    ;initialize the interrupt control register
        move.b  #$b0,ACR      ;initialize the auxiliary control register
        move.b  #$ff,CUR      ;initialize the upper preload register
        move.b  #$ff,CLR      ;initialize the lower preload register
        move.b  SPC,tmp       ;stop the timer
        move.b  #%00001000,IMR;initialize the interrupt mask register
*                             ;for timer interrupt
        move.b  STC,tmp       ;start the timer

* main program continuously print 'N's
loopt   btst.b  #2,SRB
        move.b  #N,TBB
        bra     loopt

* interrupt service routine - print 100 'I's for every interrupt
        org     $30000
inter   move.b  SPC,tmp ;stop the timer
        move.l  #100,d1
loopi   btst.b  #2 ,SRB
        beq     loopi
        move.b  #I,TBB
        dbra    d1,loopi
        move.b  STC,tmp ;restart the timer
        rte
```

8.2.5.2 Example 2 - Autovectored Timer Interrupts

```
SRB     equ     $FFFFF7F3 ;duart status register
TBB     equ     $FFFFF7F7 ;duart transmission buffer
ICR     equ     $FFFFFFFA ;cpu interrupt control register
IMR     equ     $FFFFF7EB ;duart interrupt mask register
IVR     equ     $FFFFF7F9 ;duart interrupt vector register
ACR     equ     $FFFFF7E9 ;duart auxiliary control register
```

```
CUR     equ     $FFFFF7EF ;timer counter upper preload register
CLR     equ     $FFFFF7ED ;timer counter lower preload register
STC     equ     $FFFFF7FD ;timer start counter command register
SPC     equ     $FFFFF7FF ;timer stop counter command register
tmp     equ     $31000
N       equ     $4E
I       equ     $49

        org     $20000
* initialize the DUART for level 4 interrupt
        move.l  #inter,$70 ;initialize vector table
        move.b  #28,IVR    ;initialize DUART interrupt vector register
        move.w  #$2300,sr  ;initialize processor status register

* initialize the counter
        move.b  #$b0,ACR    ;initialize the auxiliary control register
        move.b  #$ff,CUR ;initialize the timer counter - high byte
        move.b  #$ff,CLR ;initialize the timer counter - low byte

* enable DUART to interrupt when the timer is ready
        move.b  SPC,tmp      ;stop the counter
        move.b  #%00001000,IMR ;initialize for counter interrupt
        move.b  STC,tmp      ;start the timer

* main loop of the program - continously print out the character 'N'.
loopt   btst.b  #2,SRB
        move.b  #N,TBB
        bra     loopt

        org     $30000
inter   move.b  SPC,tmp ;stop the counter
        move.l  #100,d1   ;print 'I' 100 times
loopi   btst.b  #2,SRB
        beq     loopi
        move.b  #I,TBB
        dbra    d1,loopi
        move.b  STC,tmp ;start the timer
        rte
```

8.3 Parallel Interface/Timer

The Ultragizmo board has a Parallel Interface/Timer (PIT) port, used for **parallel I/O** (as opposed to the **serial I/O** on the DUART). The PIT is a 40-pin I/O connector controlled by the CFPGA. The location of the PIT on the Ultragizmo board is shown in Figure 42 (on page 92). The PIT provides two 8-bit I/O ports labelled PA and PB and access to the 16-bit DUART Timer output of MC68306. All PIT signals are fully available to the user. Figure 45 shows the connections of a PIT and its pin assignment.

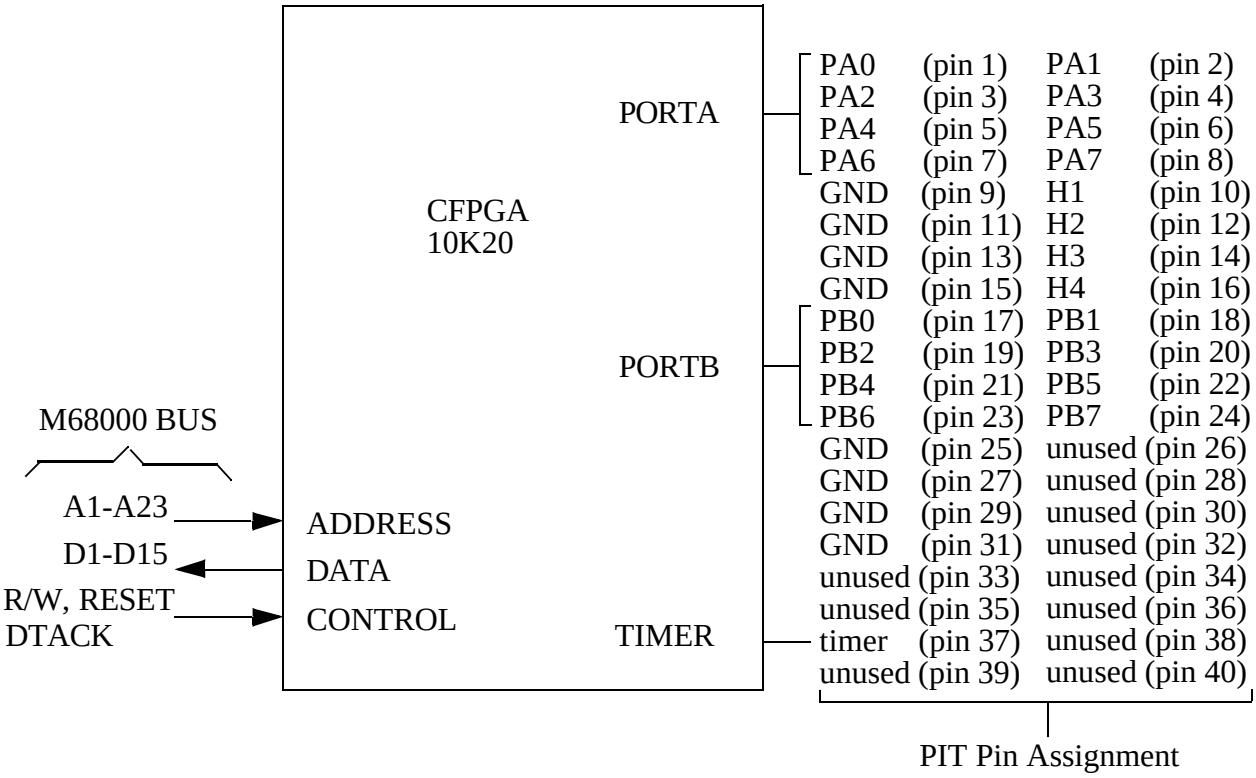

Figure 45 - *Pin Connections for the Parallel Interface/Timer*

8.3.1 PIT Overview

The primary function of the PIT is to provide an interface that allows the CPU to communicate with other logic circuits by means of registers that drive or receive logical values on the pins of the PIT. The PIT contains two separate ports, each of which contains 8 Boolean signals that can be driven or read. Each logic signal is connected to a pin on the PIT. The names of the pins are PA0 through PA7 and PB0 through PB7.

A simplified logic diagram of the circuitry connected to each pin is shown in Figure 46. The figure shows the logic connected to pin PA0. Each pin is controlled by a control bit, called the data direction bit, which controls whether the pin is an input or output to the PIT. This direction

bit is stored in the direction register for that port (in this case PADDR0). The PADDR register contains 8 bits, one for each of PA0-7. Writing a 1 to a bit in PADDR causes the corresponding pin to be an output. Writing a 0 causes the pin to be an input. If the pin is configured to be an output, a logical high or low can be asserted on that line by writing a 1 or 0 respectively into the corresponding bit in the data register for the port, called PADOR. If the bit is configured to be an input, the processor can determine the state of the pin by reading and testing a bit in the input port PADI. A pull-up resistor connected to the pin ensures that unconnected inputs are always read as 1.

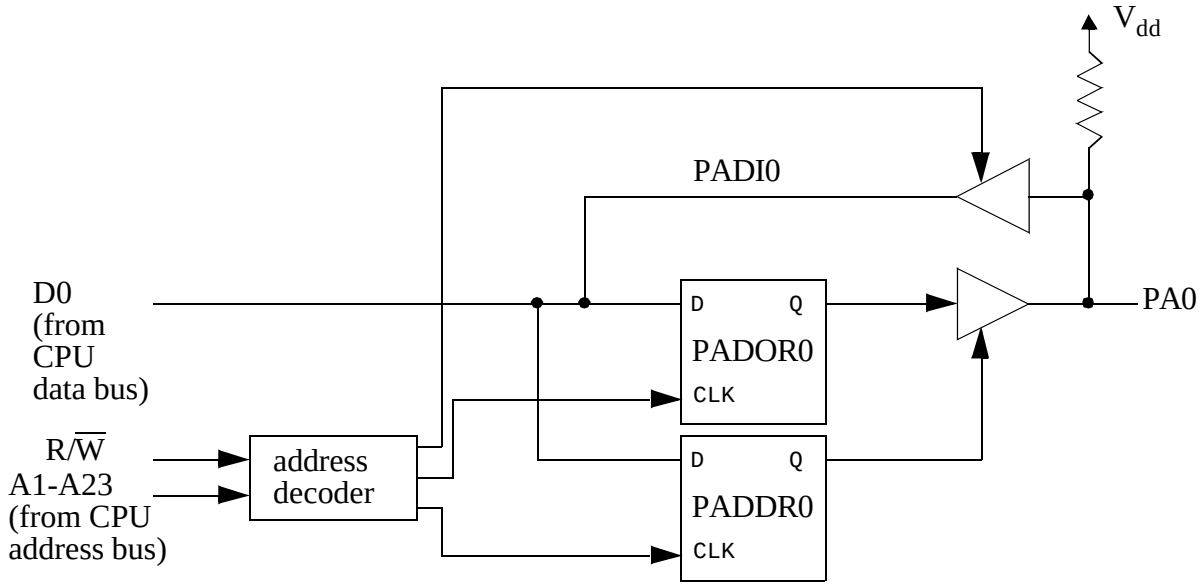

Figure 46 - *Port Logic for Parallel Interface/Timer*

It is possible to use some pins as outputs and some pins as inputs within a single port. For example, we could use pins PA0-5 as outputs and pins PA6-7 as inputs by writing $3F to PADDR. We could drive logic 1 on PA0 and PA3, and 0 on PA1, PA2, PA4, and PA5 by writing $09 to PADOR. The external state of pins PA6 and PA7 could be determined by testing bits 6 and 7 of PADI.

External devices can generate level 1 vectored/autovectored interrupts or level 5 autovectored interrupts through the PIT. A procedure for enabling the level 1 or level 5 interrupt is described in Section 8.2 on page 100. Pins H1-H4 on the PIT are used for interrupt purposes. When the level 1 interrupt is enabled, H1 acts as the interrupt request signal, and H2 acts as the interrupt acknowledge signal. A level 1 interrupt is triggered when H1 is pulled down to ground. When the level 5 autovectored interrupt is enabled, any H1, H2, H3, or H4 pin acts as the interrupt request signal. A level 5 interrupt is triggered when any of these four pins has an edge transition (see Table 10 for transition types).

The PIT also allows external devices to access the 16-bit DUART timer of the MC68306. Pin 37 of the PIT is connected to the output of the DUART Timer, which is referred to as the OP3 pin in the MC68306 User's Manual. The details of configuring and using the DUART Timer can be found in the MC68306 User's Manual on page 6-16 to 6-17. A sample program is given in Section 8.2.5 on page 107.

Table 13 gives a summary of the PIT registers and their addresses on the Ultragizmo board.

Address	REGISTER	Name
$00C10000	Port A Data Output Register	PADOR
$00C10001	Port B Data Output Register	PBDOR
$00C10002	Port A Data Direction Register	PADDR
$00C10003	Port B Data Direction Register	PBDDR
$00C10004	Port A Data Input	PADI
$00C10005	Port B Data Input	PBDI
$00C20000	CFPGA Interrupt Configuration Register	CFPGA_ICR
$FFFFF7E9	DUART Auxiliary Control Register	DUACR
$FFFFF7ED	DUART Counter/Timer Upper Register	DUCTUR
$FFFFF7EF	DUART Counter/Timer Lower Register	DUCTLR
$FFFFF7FD	DUART Start Counter Command	DUSTART
$FFFFF7FF	DUART Stop Counter Command	DUSTOP

Table 13 - *Register Addresses for the PIT*

8.3.2 Using the PIT to Understand Memory-Mapped I/O

The following exercise is intended to show, in a direct way, the link between memory locations and the physical world. Here, the PIT is used to access values put onto the M68000 bus. For this exercise, you will need a cable, a proto-board connector, a proto-board, a logic probe and one wire. Make sure that the cable has connectors that have “key” plugs on them so that there is only one way to plug it in.

1. Plug the proto-board in and turn it on. Check that the logic probe works by touching the probe to +5V and ground. The “high” and “low” lights should come on respectively.
2. Plug the cable into connector PIT on the Ultragizmo board. See Figure 42 (on page 92) for the position of the PIT Connector.
3. Plug the other end of the cable into the proto-board connector (if it isn’t already) and then into the proto-board. Make sure it is straddling a valley in the board. The pins can now be connected to by plugging a wire into any one of the vertically adjacent holes in the proto board.
4. Hook up pin #11 of the connector to ground on the proto- board. This makes the ground on the proto-board the same as the ground on the Ultragizmo board. Pin number 1 of the proto- board connector is indicated by a small triangle. The next pin on the same side is #3, and so on. Pin #2 is opposite Pin #1, and Pin #4 is opposite Pin #3.
5. Use the monitor **block fill** command (**bf**) to set up the control registers and registers for Port A of the PIT in the following way. (We configure it to be an output, and to output $f0)
    i. Put $ff in memory location $c10002 (PADDR - set data direction to be all out)

ii. Put $f0 in memory location $c10000 (PADOR - output 11110000 through port A)

6. Using the logic probe, measure the binary value on each of the pins PA0 (Proto-board connector pin #1), PA1 (#2), PA2 (#3)... PA7 (#8). These can be measured by touching the solder traces on the board that the connector is soldered to.

   Make sure that the value put into the PADOR ($f0 =$11110000_2$) is what you read with the logic probe.

7. Change the value in PADOR, and make sure that the output logic probe values change correctly.

8.3.3 Creating a Second PIT on the SFPGA

Some projects might require a second PIT. This can be created on port **SFPGA_CON40** using the 10K70 SFPGA. The second PIT has the same functionality as the regular PIT, with the following two exceptions. First, the PIT is not connected to the DUART timer. Second, the PIT can only be configured to generate level 6 autovectored interrupts since this is the interrupt level supported by the SFPGA.

To create the second PIT, the user needs to use MAX+plusII (see the FPGA downloading instructions from the FPGA tutorial in Chapter 3) to download the *X:\DEMOS\PIT.SOF* file onto the Ultragizmo board. This file creates the second pit on the SFPGA (See *X:\DEMOS\README* for a more detailed description). The addresses for this PIT are listed below.

Address	REGISTER	Name
00C20000	CFPGA Interrupt Configuration Register	CFPGA_ICR
00C50000	Port A Data Output Register	PADOR2
00C50001	Port B Data Output Register	PBDOR2
00C50002	Port A Data Direction Register	PADDR2
00C50003	Port B Data Direction Register	PBDDR2
00C50004	Port A Data Input	PADI2
00C50005	Port B Data Input	PBDI2
00C50006	Port Vector Register	PVR2

Table 14 - *Creating a Second PIT on the SFPGA*

The second PIT uses $\overline{\text{IRQ6}}$. To enable interrupts on the second PIT you have to write $06 into the CFPGA_IVR (location $c20000), and write $06 into the PVR2 (location $c50006). The *X:\DEMOS\INTER6.SREC* file can be used to test the interrupts. An interrupt is triggered when any of H1, H2, H3, H4 is low. To further test the second PIT, you can use *X:\DEMOS\MOTSFPGA.SREC* with the LEGO controller, starting the program at address $20000 and following the menu instructions.

8.4 CPU Expansion Bus

The Ultragizmo board has a buffered version of the main CPU bus available through connector **M68000_BUS**. This section contains a brief description of the expansion bus signals. All of the CPU address bus, the data bus and the bus arbitration control are available on the **M68000_BUS**, which is a 60-pin connector. Three interrupt control signals are provided for interrupt requests. The following is a list of the available signals.

- Address Bus (A23-A1): A full 23-bit bidirectional three-state bus is provided. The address bus is buffered with non-inverting buffers.
- Data Bus (D15-D0): A full 16-bit bidirectional three-state bus is provided. The data bus is buffered with non-inverting buffers.
- Asynchronous Control ($\overline{AS}$, $\overline{LDS}$, $\overline{UDS}$, $\overline{DTACK}$, R/$\overline{W}$): The asynchronous control signals are provided as bidirectional buffered three-state signals.
- Bus Arbitration control ($\overline{BR}$, $\overline{BGACK}$, $\overline{BG}$): The Bus Request signal $\overline{BR}$ can be asserted to request an off board bus request. The bus grant $\overline{BG}$ is asserted when the CPU grants the bus to the external request. $\overline{BGACK}$ is used to acknowledge the $\overline{BG}$ signal
- Interrupt Control ($\overline{IRQ2}$, $\overline{IACK2}$, $\overline{AUTO3}$): Three interrupt control signals are provided. $\overline{AUTO3}$ can be used to request an autovectored level 3 interrupt. $\overline{IRQ2}$ and $\overline{IACK2}$ can be used to request both autovectored or vectored level 2 interrupts.
- System Control ($\overline{RESET}$, CLK): The reset signal $\overline{RESET}$ is a bidirectional three-state signal. The clock signal CLK is a buffered version of the 16.67 MHz CPU clock.

Table 15, “Pin Assignment for CPU Expansion Bus (M68000_BUS),” on page 116 lists the signals of the CPU Expansion Bus with their pin locations.

1	-	GND	2	-	CLK
3	-	GND	4	-	A1
5	-	A2	6	-	A3
7	-	A4	8	-	A5
9	-	A6	10	-	A7
11	-	A8	12	-	A9
13	-	A10	14	-	A11
15	-	A12	16	-	A13
17	-	A14	18	-	A15
19	-	A16	20	-	A17
21	-	A18	22	-	A19
23	-	A20	24	-	A21
25	-	A22	26	-	A23
27	-	R/$\overline{W}$	28	-	$\overline{DTACK}$
29	-	GND	30	-	$\overline{AS}$
31	-	GND	32	-	$\overline{UDS}$
33	-	GND	34	-	$\overline{LDS}$
35	-	GND	36	-	$\overline{BR}$
37	-	GND	38	-	$\overline{BG}$
39	-	GND	40	-	$\overline{BGACK}$
41	-	$\overline{RESET}$	42	-	$\overline{IACK2}$
43	-	$\overline{AUTO3}$	44	-	$\overline{IRQ2}$
45	-	D0	46	-	D1
47	-	D2	48	-	D3
49	-	D4	50	-	D5
51	-	D6	52	-	D7
53	-	D8	54	-	D9
55	-	D10	56	-	D11
57	-	D12	58	-	D13
59	-	D14	60	-	D15

Table 15 - *Pin Assignment for CPU Expansion Bus (M68000_BUS)*

8.5 SFPGA

The SFPGA is an Altera FLEX 10K70 device. It is used for fast prototyping of M68000 peripherals. The pins of the SFPGA are connected to the M68000 bus and several external connectors (**SFPGA_DIGITAL**, **SFPGA_CON40**, **SFPGA_CON60**, **SFPGA_LOGIC**) shown in Figure 42 (on page 92). This section describes the pin connections to the SFPGA in detail.

Unused pins on the Altera FLEX 10K70 become outputs and are driven low by default. To ensure that unused SFPGA pins do not drive the M68000 bus, all unused pins connected to the bus have to be defined as inputs or high impedance outputs.

To simplify the pin assignment for your design, two wrapper files, *wrapper.vhd* and *wrapper.acf*, are provided in the directory */cad2/ultragizmo/FLabs/* on the *ugsparc* system. Pin assignment information has already been entered into these files. You can use these wrapper files as your top level design files. Any output signals that are not used in the wrapper files should be reassigned as input pins or driven to high impedance.

The following sections summarize the pin assignments to the SFPGA chip. Tables in these sections contain three fields. The first field lists the names of the signals, the second field lists the names of the corresponding SFPGA pins, and the third field contains the VHDL names in the wrapper files.

8.5.1 LEDs

There are 16 **LED**s on the Ultragizmo board, just above the SFPGA (See Figure 42 for their locations). The **LED**s are connected to the SFPGA and can be used for debugging purposes. Table 16 lists the pin assignments for the **LED**s. See Figure 42 for the numbering of the **LED**s. **LED**s are active low devices. To turn on an **LED** the corresponding signal has to be driven low and vice versa.

LED Name	SFPGA Pin Name	VHDL Signal Name
$\overline{\text{LED1}}$	A9	led(0)
$\overline{\text{LED2}}$	AT42	led(1)
$\overline{\text{LED3}}$	D32	led(2)
$\overline{\text{LED4}}$	AE37	led(3)
$\overline{\text{LED5}}$	L1	led(4)
$\overline{\text{LED6}}$	AP40	led(5)
$\overline{\text{LED7}}$	AM38	led(6)
$\overline{\text{LED8}}$	B34	led(7)

Table 16 - *Pin Assignment for the LEDs*

LED Name	SFPGA Pin Name	VHDL Signal Name
$\overline{\text{LED9}}$	B12	led(8)
$\overline{\text{LED10}}$	A11	led(9)
$\overline{\text{LED11}}$	AR43	led(10)
$\overline{\text{LED12}}$	AC43	led(11)
$\overline{\text{LED13}}$	G15	led(12)
$\overline{\text{LED14}}$	E33	led(13)
$\overline{\text{LED15}}$	AL3	led(14)
$\overline{\text{LED16}}$	Y40	led(15)

Table 16 - *Pin Assignment for the LEDs*

8.5.2 Hex Displays

There are two banks of hex displays on the Ultragizmo board. Each bank contains two hex displays. The hex displays are located below the SFPGA on the board (See Figure 42 for their location; note the ordering of the displays). These four hex displays are connected to the SFPGA. They can be used as digital displays in applications like digital watches or microprocessors. Table 17, Table 18, Table 19, and Table 20 list the pin assignments for the hex displays. Figure 47 shows the naming convention for the display segments. Hex displays are active low devices. To turn on a hex display segment, the corresponding signal has to be driven low.

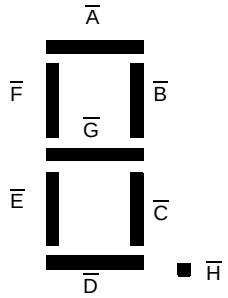

Figure 47 - *Hex Display Segment Naming Convention*

Hex Display Signal Name	SPGA Pin Name	VHDL Signal Name
$\overline{\text{A}}$	W43	hex0(6)
$\overline{\text{B}}$	W37	hex0(5)
$\overline{\text{C}}$	V38	hex0(4)

Table 17 - *Pin Assignment for Hex Display 0*

Hex Display Signal Name	SPGA Pin Name	VHDL Signal Name
$\overline{D}$	U43	hex0(3)
$\overline{E}$	L37	hex0(2)
$\overline{F}$	AL37	hex0(1)
$\overline{G}$	A13	hex0(0)
$\overline{H}$	AN7	hex0(7)

Table 17 - *Pin Assignment for Hex Display 0*

Hex Display Signal Name	SPGA Pin Name	VHDL Signal Name
$\overline{A}$	AA37	hex1(6)
$\overline{B}$	G39	hex1(5)
$\overline{C}$	H38	hex1(4)
$\overline{D}$	AW1	hex1(3)
$\overline{E}$	AR5	hex1(2)
$\overline{F}$	F42	hex1(1)
$\overline{G}$	E21	hex1(0)
$\overline{H}$	Y42	hex1(7)

Table 18 - *Pin Assignment for Hex Display 1*

Hex Display Signal Name	SPGA Pin Name	VHDL Signal Name
$\overline{A}$	AY28	hex2(6)
$\overline{B}$	AA7	hex2(5)
$\overline{C}$	Y4	hex2(4)
$\overline{D}$	D10	hex2(3)
$\overline{E}$	AV28	hex2(2)
$\overline{F}$	AY14	hex2(1)
$\overline{G}$	Y6	hex2(0)
$\overline{H}$	BC29	hex2(7)

Table 19 - *Pin Assignment for Hex Display 2*

Hex Display Signal Name	SPGA Pin Name	VHDL Signal Name
$\overline{A}$	F12	hex3(6)
$\overline{B}$	AY18	hex3(5)
$\overline{C}$	W1	hex3(4)
$\overline{D}$	AW29	hex3(3)
$\overline{E}$	G13	hex3(2)
$\overline{F}$	BC31	hex3(1)
$\overline{G}$	W7	hex3(0)
$\overline{H}$	BB30	hex3(7)

Table 20 - *Pin Assignment for Hex Display 3*

8.5.3 SFPGA_DIGITAL Connector

SFPGA_DIGITAL is a 40-pin connector directly connected to the SFPGA. It is located on the bottom of the Ultragizmo board just to the left of the PIT (see Figure 42 for details). The **SFPGA_DIGITAL** connector is pin to pin compatible with the protoboard 40-pin digital connector (Section 9.1). The protoboard can easily be made available to the SFPGA by connecting these two connectors using a 40-pin ribbon cable. Table 21 lists the pin assignment for the 40-pin **SFPGA_DIGITAL** connector.

Pin	SFPGA Pin Name	VHDL Signal Name	Pin	SFPGA Pin Name	VHDL Signal Name
1	BC9	sfpga_digital(0)	2	AW23	sfpga_digital(1)
3	AU15	sfpga_digital(2)	4	A15	sfpga_digital(3)
5	AU13	sfpga_digital(4)	6	BC35	sfpga_digital(5)
7	AN43	sfpga_digital(6)	8	R1	sfpga_digital(7)
9	AM40	sfpga_digital(8)	10	AV22	sfpga_digital(9)
11	AL43	sfpga_digital(10)	12	F20	sfpga_digital(11)
13	AJ37	sfpga_digital(12)	14	BB34	sfpga_digital(13)
15	F16	sfpga_digital(14)	16	T4	sfpga_digital(15)
17	---	GND	18	AU21	sfpga_digital(16)
19	AK40	sfpga_digital(17)	20	---	GND
21	AM42	sfpga_digital(18)	22	AY32	sfpga_digital(19)
23	AN39	sfpga_digital(20)	24	AV30	sfpga_digital(21)
25	AV42	sfpga_digital(22)	26	T2	sfpga_digital(23)
27	AY34	sfpga_digital(24)	28	U5	sfpga_digital(25)

Table 21 - *Pin Assignment for the SFPGA_DIGITAL Connector*

29	AW33	sfpga_digital(26)	30	BC21	sfpga_digital(27)
31	P2	sfpga_digital(28)	32	C19	sfpga_digital(29)
33	F14	sfpga_digital(30)	34	BB32	sfpga_digital(31)
35	---	GND	36	U1	sfpga_digital(32)
37	---	GND	38	---	GND
39	---	GND	40	BA31	sfpga_digital(33)

Table 21 - *Pin Assignment for the SFPGA_DIGITAL Connector*

8.5.4 SFPGA_CON40 Connector

SFPGA_CON40 is a 40-pin connector located right above the LEDs (see Figure 42 for details). The **SFPGA_CON40** is directly connected to the SFPGA. This is a versatile connector which can be used for a variety of applications. Table 22 lists the pin assignment for **SFPGA_CON40**. See Section 8.3.3 for instructions on how to create a PIT using this connector.

Pin	SFPGA Pin Name	VHDL Signal Name	Pin	SFPGA Pin Name	VHDL Signal Name
1	L7	sfpga_con40(0)	2	B2	sfpga_con40(1)
3	BA41	sfpga_con40(2)	4	G43	sfpga_con40(3)
5	AV38	sfpga_con40(4)	6	F28	sfpga_con40(5)
7	B4	sfpga_con40(6)	8	BA43	sfpga_con40(7)
9	---	GND	10	E7	sfpga_con40(8)
11	---	GND	12	AY42	sfpga_con40(9)
13	---	GND	14	E29	sfpga_con40(10)
15	---	GND	16	K38	sfpga_con40(11)
17	G9	sfpga_con40(12)	18	AU39	sfpga_con40(13)
19	D30	sfpga_con40(14)	20	AW43	sfpga_con40(15)
21	F6	sfpga_con40(16)	22	AT38	sfpga_con40(17)
23	A5	sfpga_con40(18)	24	AR37	sfpga_con40(19)
25	---	GND	26	E9	sfpga_con40(20)
27	---	GND	28	AR39	sfpga_con40(21)
29	---	GND	30	C31	sfpga_con40(22)
31	---	GND	32	AK38	sfpga_con40(23)
33	G11	sfpga_con40(24)	34	AP39	sfpga_con40(25)
35	B32	sfpga_con40(26)	36	AH40	sfpga_con40(27)
37	E11	sfpga_con40(28)	38	AN37	sfpga_con40(29)
39	B8	sfpga_con40(30)	40	AU43	sfpga_con40(31)

Table 22 - *Pin Assignment for the SFPGA_CON40 Connector*

8.5.5 SFPGA_LOGIC Connector

The **SFPGA_LOGIC** connector is reserved for the logic analyzers available in the lab. For instructions on how to connect the logic analyzers to this port, please consult Fred Aulich's web-site at *http://www.eecg.toronto.edu/~aulich.* This 40-pin connector is located on the right side of the Ultragizmo board. Table 23 summarizes the pin assignments for the SFPGA_LOGIC connector.

Pin	SFPGA Pin Name	VHDL Signal Name	Pin	SFPGA Pin Name	VHDL Signal Name
1	---	GND	2	AU23	sfpga_logic(0)
3	AV24	sfpga_logic(1)	4	---	GND
5	AB6	sfpga_logic(2)	6	---	GND
7	D8	sfpga_logic(3)	8	---	GND
9	BB26	sfpga_logic(4)	10	---	GND
11	BC27	sfpga_logic(5)	12	---	GND
13	AY26	sfpga_logic(6)	14	---	GND
15	AB4	sfpga_logic(7)	16	---	GND
17	AB2	sfpga_logic(8)	18	---	GND
19	---	GND	20	---	GND
21	---	GND	22	---	GND
23	AW27	sfpga_logic(9)	24	---	GND
25	AV26	sfpga_logic(10)	26	---	GND
27	AA1	sfpga_logic(11)	28	---	GND
29	F10	sfpga_logic(12)	30	---	GND
31	AA5	sfpga_logic(13)	32	---	GND
33	A7	sfpga_logic(14)	34	---	GND
35	BB28	sfpga_logic(15)	36	---	GND
37	AV14	sfpga_logic(16)	38	---	GND
39	---	GND	40	Y2	sfpga_logic(17)

Table 23 - *Pin Assignment for the SFPGA_LOGIC Connector*

8.5.6 SFPGA_CON60 Connector

The **SFPGA_CON60** is a 60-pin connector located on the right side of the Ultragizmo board (see Figure 42 for details). Besides application specific functions, this connector is pin to pin compatible with the **M68000_BUS** connector. Since the **M68000_BUS** contains some bus signals that are not available to the SFPGA through the direct connection (Section 8.5.7), additional M68000 bus signals can be made available to the SFPGA by connecting the **SFPGA_CON60** and the **M68000_BUS** using a 60-pin ribbon connector.

Pin	SFPGA Pin Name	VHDL Signal Name	Pin	SFPGA Pin Name	VHDL Signal Name
1	---	GND	2	R5	sfpga_con60(0)
3	---	GND	4	BA35	sfpga_con60(1)
5	F22	sfpga_con60(2)	6	BB24	sfpga_con60(3)
7	BC37	sfpga_con60(4)	8	E17	sfpga_con60(5)
9	A17	sfpga_con60(6)	10	BC25	sfpga_con60(7)
11	AV34	sfpga_con60(8)	12	AU33	sfpga_con60(9)
13	G19	sfpga_con60(10)	14	E23	sfpga_con60(11)
15	R7	sfpga_con60(12)	16	P6	sfpga_con60(13)
17	AW35	sfpga_con60(14)	18	AU25	sfpga_con60(15)
19	BB36	sfpga_con60(16)	20	AY36	sfpga_con60(17)
21	F24	sfpga_con60(18)	22	D42	sfpga_con60(19)
23	M4	sfpga_con60(20)	24	BB38	sfpga_con60(21)
25	A25	sfpga_con60(22)	26	BC39	sfpga_con60(23)
27	V36	sfpga_con60(24)	28	AU35	sfpga_con60(25)
29	---	GND	30	AJ1	sfpga_con60(26)
31	---	GND	32	B26	sfpga_con60(27)
33	---	GND	34	K2	sfpga_con60(28)
35	---	GND	36	J1	sfpga_con60(29)
37	---	GND	38	AW37	sfpga_con60(30)
39	---	GND	40	BA33	sfpga_con60(31)
41	M6	sfpga_con60(32)	42	AY38	sfpga_con60(33)
43	A27	sfpga_con60(34)	44	AU31	sfpga_con60(35)
45	L5	sfpga_con60(36)	46	H2	sfpga_con60(37)
47	AV32	sfpga_con60(38)	48	BA39	sfpga_con60(39)
49	BC41	sfpga_con60(40)	50	C27	sfpga_con60(41)
51	G1	sfpga_con60(42)	52	BB40	sfpga_con60(43)
53	C7	sfpga_con60(44)	54	BC43	sfpga_con60(45)
55	B28	sfpga_con60(46)	56	H42	sfpga_con60(47)
57	J5	sfpga_con60(48)	58	BB42	sfpga_con60(49)
59	A29	sfpga_con60(50)	60	U41	sfpga_con60(51)

Table 24 - *Pin Assignment for the SFPGA_CON60 Connector*

8.5.7 M68000 Bus to SFPGA Connections

Several SFPGA pins are directly connected to the M68000 bus. Table 25 lists the names of the M68000 bus signals, the SFPGA pins connected to the signals, and the VHDL signal names used in the wrapper file. The **MC68306 User's Manual** gives a detailed description of the signals.

M68000 Bus Signal Name	SFPGA Pin Name	VHDL Signal Name	M68000 Bus Signal Name	SFPGA Pin Name	VHDL Signal Name
A1	AK4	address(1)	A16	AK6	address(16)
A2	J7	address(2)	A17	E1	address(17)
A3	B20	address(3)	A18	T6	address(18)
A4	AJ7	address(4)	A19	AM4	address(19)
A5	A43	address(5)	A20	J37	address(20)
A6	B38	address(6)	A21	F38	address(21)
A7	D36	address(7)	A22	F36	address(22)
A8	B42	address(8)	A23	AD42	address(23)
A9	AK2	address(9)	$\overline{AS}$	L41	as
A10	AL1	address(10)	$\overline{BERR}$	M38	berr
A11	D16	address(11)	$\overline{BG}$	AU29	bg
A12	M42	address(12)	$\overline{BGACK}$	AC1	bgack
A13	P40	address(13)	$\overline{BR}$	BC13	br
A14	R39	address(14)	CLK	AY22	clk
A15	B10	address(15)	D0	AL7	data(0)
D1	E43	data(1)	D14	AN1	data(14)
D2	D40	data(2)	D15	AM2	data(15)
D3	V40	data(3)	FC2	AP4	fc2
D4	G41	data(4)	FC1	AM6	fc1
D5	AT2	data(5)	FC0	AU1	fc0
D6	AN5	data(6)	$\overline{DTACK}$	AG3	dtack
D7	V6	data(7)	$\overline{HALT}$	AJ5	halt
D8	AR1	data(8)	$\overline{IACK6}$	D12	iacksf
D9	U39	data(9)	$\overline{IRQ6}$	AV20	irqsf
D10	T42	data(10)	$\overline{LDS}$	J43	lds
D11	T38	data(11)	$\overline{RESET}$	AH6	reset
D12	R43	data(12)	R/$\overline{W}$	AH2	rw
D13	AP2	data(13)	$\overline{UDS}$	AH4	uds

Table 25 - *Pin Assignment for the M68000 Bus Signals*

8.5.8 SRAM to SFPGA Connections

Two SRAM chips are attached to the SFPGA. Each chip has the capacity of 16x256k bits. On the board the chips are configured to provide 32x256k bits of memory. Table 26 summarizes the pin assignment for the SRAM. The two SRAMs share the same address signals, but each has separate control and data signals. The data signals for SRAM 1 are named **data_l_0** to **data_l_15**. The control signals are **enable1**, **ld1**, **ud1**, **we1**, **oe1**. The data signals for SRAM 2 are named **data_h_0** to **data_h_15**. The control signals are **enable2**, **ld2**, **ud2**, **we2**, **oe2**. See the **Samsung KM6164002 SRAM User's Manual** for more details on the SRAM signals.

SRAM Signal Name	SFPGA Pin Name	VHDL Signal Name	SRAM Signal Name	SFPGA Pin Name	VHDL Signal Name
data_l_0	C43	sramdl(0)	data_l_1	AA39	sramdl(1)
data_l_2	AA43	sramdl(2)	data_l_3	AP6	sramdl(3)
data_l_4	AR7	sramdl(4)	data_l_5	AV2	sramdl(5)
data_l_6	Y38	sramdl(6)	data_l_7	AV6	sramdl(7)
data_l_8	AY2	sramdl(8)	data_l_9	BA1	sramdl(9)
data_l_10	BB2	sramdl(10)	data_l_11	BA3	sramdl(11)
data_l_12	N37	sramdl(12)	data_l_13	J39	sramdl(13)
data_l_14	AB42	sramdl(14)	data_l_15	AB40	sramdl(15)
data_h_0	AB38	sramdh(0)	data_h_1	AC37	sramdh(1)
data_h_2	AC39	sramdh(2)	data_h_3	AP42	sramdh(3)
data_h_4	AT6	sramdh(4)	data_h_5	AU5	sramdh(5)
data_h_6	BC1	sramdh(6)	data_h_7	AY6	sramdh(7)
data_h_8	AV8	sramdh(8)	data_h_9	BC5	sramdh(9)
data_h_10	AG43	sramdh(10)	data_h_11	AF42	sramdh(11)
data_h_12	N43	sramdh(12)	data_h_13	AD38	sramdh(13)
data_h_14	AD40	sramdh(14)	data_h_15	AE41	sramdh(15)
address0	GE43	srama(0)	address1	AF40	srama(1)
address2	AU9	srama(2)	address3	AW7	srama(3)
address4	BA5	srama(4)	address5	BB4	srama(5)
address6	BB6	srama(6)	address7	AV10	srama(7)
address8	BC7	srama(8)	address9	AY10	srama(9)
address10	AW11	srama(10)	address11	AJ37	srama(11)
address12	AH38	srama(12)	address13	AH42	srama(13)
address14	AG39	srama(14)	address15	AG41	srama(15)
address16	AF38	srama(16)	address17	AJ43	srama(17)
enable1	AG41	sram1en	enable2	BB12	sram2en
ld1	AK42	sram1ld	ld2	AV12	sram2ld

Table 26 - *Pin Assignment for SRAM Signals*

ud1	AU11	sram1ud	ud2	BB10	sram2ud
we1	AW9	sram1we	we2	AY12	sram2we
oe1	AY8	sram1oe	oe2	BC11	sram2oe

Table 26 - *Pin Assignment for SRAM Signals*

8.5.9 CODEC to SFPGA Connections

The CODEC provides audio capability to the Ultragizmo board. It is attached to the SFPGA, as well as the M68000 bus. The SFPGA can control the CODEC through three signals, **ssync**, **sclk**, and **lrsync**. Two serial lines are used to transfer data between the SFPGA and the CODEC. Line **sdout** is a serial line from the CODEC to the SFPGA. Line **sdin** is a serial line from the SFPGA to the CODEC. A shift register must be designed for the SFPGA in order to convert the input serial stream into a 32-bit word. Similarly, a shift register must be used to convert an output 32-bit word into a serial bit stream for the CODEC. Section 8.6 provides more details on the CODEC and how to configure it for use with both the M68000 and the SFPGA. The SFPGA pin assignments for the CODEC are listed in Table 27:

CODEC Signal Name	SFPGA Pin Name	VHDL Signal Name	CODEC Signal Name	SFPGA Pin Name	VHDL Signal Name
sdout	BC3	sdout_cod	ssync	C3	ssync_cod
sclk	AG1	sclk_cod	lrsync	AW15	lrsync_cod
sdin	C1	sdin_cod			

Table 27 - *Pin Assignment for CODEC Signals*

8.5.10 Programmable Clock to SFPGA Connections

The output of the Cypress ICD2053B programmable clock chip is connected to pin D22 of the SFPGA. See Section 8.7 on page 130 for details on how to program the output clock frequency of the programmable clock chip.

8.6 The CODEC and Audio I/O

The CODEC (COder-DECoder) allows the input and output of audio signals via jacks on the Ultragizmo board (see Figure 42 on page 92 for the locations of the jacks; the default microphone jack is DC coupled; the AC coupled jack can be used by configuring a command register on the CFPGA; see Fred Aulich for instructions). The CODEC has a sampling rate of 48 kHz. Two 16-bit channels are available for stereo sound. The CODEC has three modes of operation, the receiver mode, the transmitter mode and the SFPGA mode. These three modes are controlled by the CODEC mode control register (CONCNTL). The CONCNTL register is an 8-bit register located at memory location $c80000 on the M68000 bus. The MC68306 processor can set the CODEC into receiver, transmitter, or SFPGA mode by writing $01, $02, or $f0 into the CONCNTL register, respectively.

In either receiver mode and transmitter mode, the CODEC is controlled by the processor. M68000 programs can communicate with the CODEC using the following set of registers:

CODCNTL	$c80000	8-bit CODEC mode control register
CODSTAT	$c80002	8-bit CODEC status register
CODRECL	$c80010	16-bit data received from input channel one of the CODEC
CODRECH	$c80012	16-bit data received from input channel two of the CODEC
CODTRSL	$c80014	16-bit data send to output channel one of the CODEC
CODTRSH	$c80016	16-bit data send to output channel two of the CODEC

Table 28 - *CODEC Registers*

In the receiver mode, the CODEC samples both audio input channels and places the sampled values into two 16-bit registers, CODRECL and CODRECH. Once the CODEC completes one sampling cycle, it informs the processor by setting the 8-bit CODEC status register, CODSTAT, to $04. The CODEC then waits until the processor reads CODRECL and CODRECH (this should be done as soon as possible). After reading the CODRECL and CODRECH, the processor must reset the CODSTAT and CODCNTL registers to $00. The CODEC then proceeds to the next sampling cycle.

In the transmitter mode, the CODEC reads the values of two 16-bit registers, CODTRSL and CODTRSH, converting them into left and right audio signals, respectively. The processor initiates a transmission by writing to the CODTRSL and CODTRSH registers and then writing $02 into CODCNTL. Once the transmission is complete, the CODEC will set the status register, CODSTAT, to $03. The processor can thenstart the next transmission.

The following M68000 code segment reads from the CODEC input channels and immediately writes the information to the output channels.

```
CODRECL    equ $c80010
CODRECH    equ $c80012
CODTRSL    equ $c80014
CODTRSH    equ $c80016
CODSTAT    equ $c80002
CODCNTL    equ $c80000

         org $20000

loop1    move.b  #$01,CODCNTL " set for read
loop     cmpi.b  #$04,CODSTAT " wait for codec data
         bne     loop
         move.w  CODRECL,CODTRSL
         move.w  CODRECH,CODTRSH
         move.b  #$00,CODSTAT
         move.b  #$00,CODCNTL

         move.b  #$02,CODCNTL " set for send
lop3     cmpi.b  #$03,CODSTAT " wait for codec data to be sent
         bne     lop3
         bra     loop1
```

In the SFPGA mode, the CODEC is accessed through the 10K70 SFPGA, as shown in Figure 48. Three control signals are used.

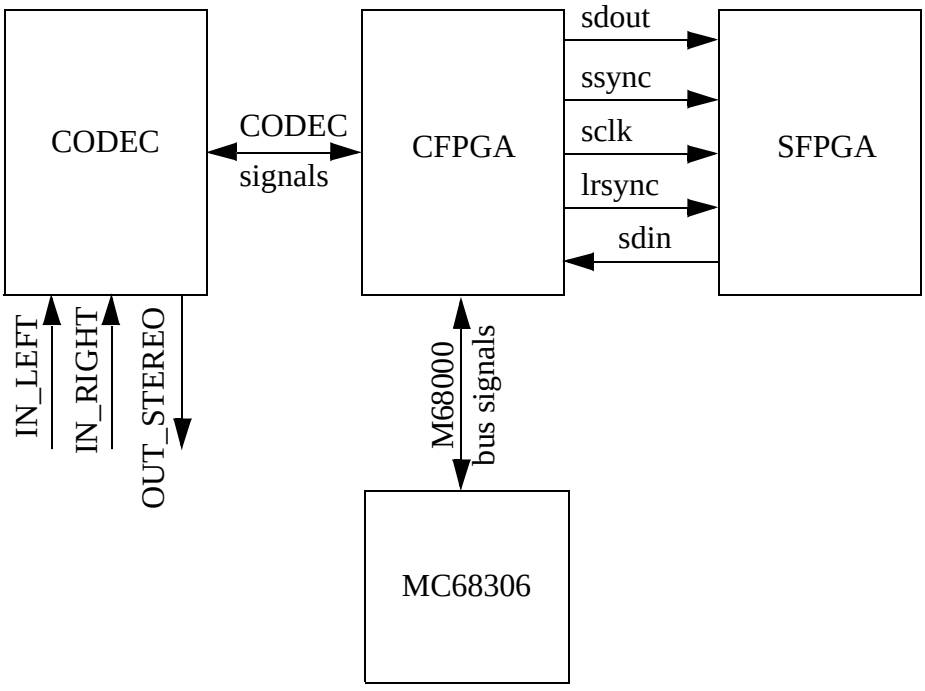

Figure 48 - *CODEC Signal Connections*

A typical read and write cycle timing diagram, in the 10K70 mode, is shown in Figure 49. The **ssync** pulse has a 48 kHz frequency, equivalent to 32 **sclk** pulses; **ssync** is low for 31 **sclk** cycles and high for one **sclk** cycle. During one **ssync** cycle, the **lrsync** signal, which indicates the chan-

nel being read or written, is high for 16 cycles (indicating an access to the left channel), and low for 16 cycles (indicating an access to the right channel). During an **ssync** period, a read or write can be performed. Writes are performed serially using the **sdout** signal on a positive **sclk** edge. Reads are performed serially using the **sdin** signal on a negative **sclk** edge. For each channel, the MSB is read or written first, and the LSB last. The first write takes place on the falling **ssync** edge and the first read takes place on the subsequent **sclk** falling edge. See Section 8.5.9 for pin locations on the SFPGA.

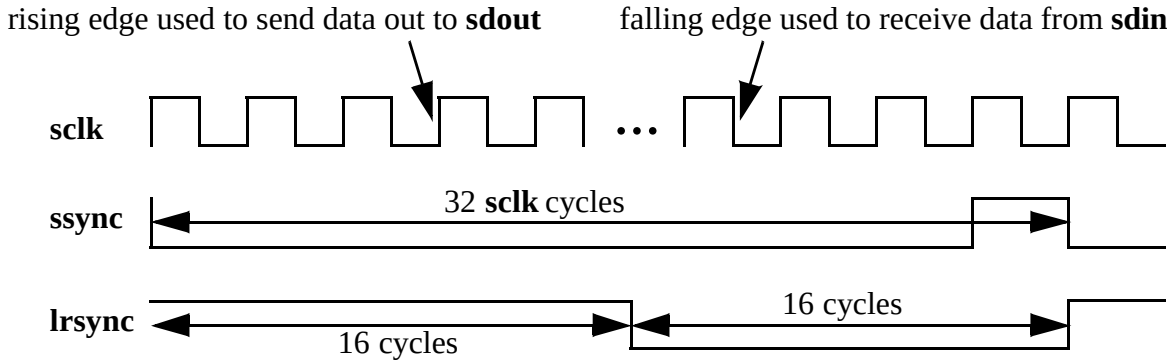

Figure 49 - *CODEC Timing Diagram*

In order to read or write data to the CODEC, two serial shift registers must be built as shown in Figure 50. One shift register converts the serial input **sdin** to a 32-bit parallel string **pdin**; the other converts a 32-bit parallel string **pdout** to the serial output **sdout**. In the figure the control logic generates the clocking for the shift registers.

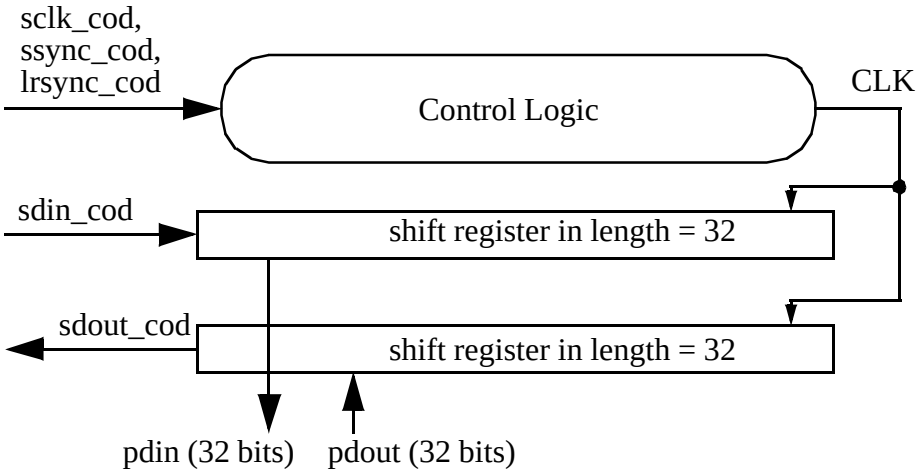

Figure 50 - *CODEC Serial-to-Parallel Converter*

More information on the CODEC can be obtained from Fred Aulich's website at *http://www.eecg.toronto.edu/~aulich*.

8.7 Programmable Clock

A Cypress ICD2053B programmable clock chip is connected to pin D22 of the SFPGA. The reference frequency for the chip is 24 MHz. The chip can be programmed to generate a clock with frequencies between 391 kHz and 90 MHz. The FBUG command **cc**, clock configure, is used to program the chip. The command takes a single argument, which is a number that specifies the desired frequency in Hz. For example, to program the clock chip to generate a 10 MHz output signal, simply type the following command at the monitor prompt (the '&' is used to indicate decimal numbers in the FBUG monitor syntax):

Ultrag > **cc &10000000**