National Centre for Radio Astrophysics

Internal Technical Report

GMRT/SERVO/PC104/006-JAN-2011

PC104 ISA Bus to GMRT SS Bus Hardware Interface Details

Giant Metrewave Radio Telescope

Tata Institute of Fundamental Research

Khodad – 410504

Document Status History

S.No	Document Status	Description
1	Document Title	PC104 ISA Bus to GMRT SS Bus Hardware Interface Details
2	Document Reference No.	GMRT/SERVO/PC104/006-DEC-2010
3	Revision	0.0
4	Date	10-January-2011
5	Prepared By	Thiyagarajan Beeman
6	Reviewed By	Servo Group Members
7	Approved By	SMEC Committee

Document Change Record

DCR No	First Issue	
Date	10-January-2011	
Document Title	PC104 ISA Bus to GMRT SS Bus Hardware Interface Details	
Document Reference No	GMRT/SERVO/PC104/006-DEC-2010	
Document Revision No	0.0	
Page No	**Paragraph**	**Reason For Change**

ACRONYM

ISA	Industry Standard Architecture
SS BUS	Station Servo computer Bus
AIO	Analog Input and Output
ENC	Encoder Card
SSC	Station Servo Computer
CPLD	Complex Programmable Logic Device
FRC	Flat Ribbon Cable
I/O	Input and Output
LED	Light Emitting Diode
CPU	Central Processing Unit
SBC	Single Board Computer
TTL	Transistor Transistor Logic
IMR	Interrupt Mask Register
CPLD	Complex Programmable Logical Device

ABSTRACT

This document details guidance for users to understand the interface between the GMRT Station Servo Computer BUS to Industry Standard ISA BUS. The first two sections of this document describe the interface signal requirements of SSBUS and ISA BUS and detail the I/O and memory addresses of both buses. The address translation logic manages the address mapping of various I/O and memory addresses used in the GMRT Servo Computer and ISA architecture. The signal consistencies between the two ends of the interface are managed with the help Complex Programmable Logic Device (CPLD). The final part of the document discusses the schematics of the interface hardware, logical equations, and technical specification of interface hardware.

Contents

List of Figures

List of Tables

1. Introduction

The PC104 single board computer (SBC) replaces the existing CPU and RTC in the station servo computer (SSC) bin and rest of the cards in the SSC bin are kept as it is. (i.e. Encoder board, Discrete input board, analog I/O board, TTL and Relay boards). The interface between the PC104 ISA bus and the SS bus are made through the ISA BUS to SSBUS interface card. The block diagram of this interface as shown in Figure.1

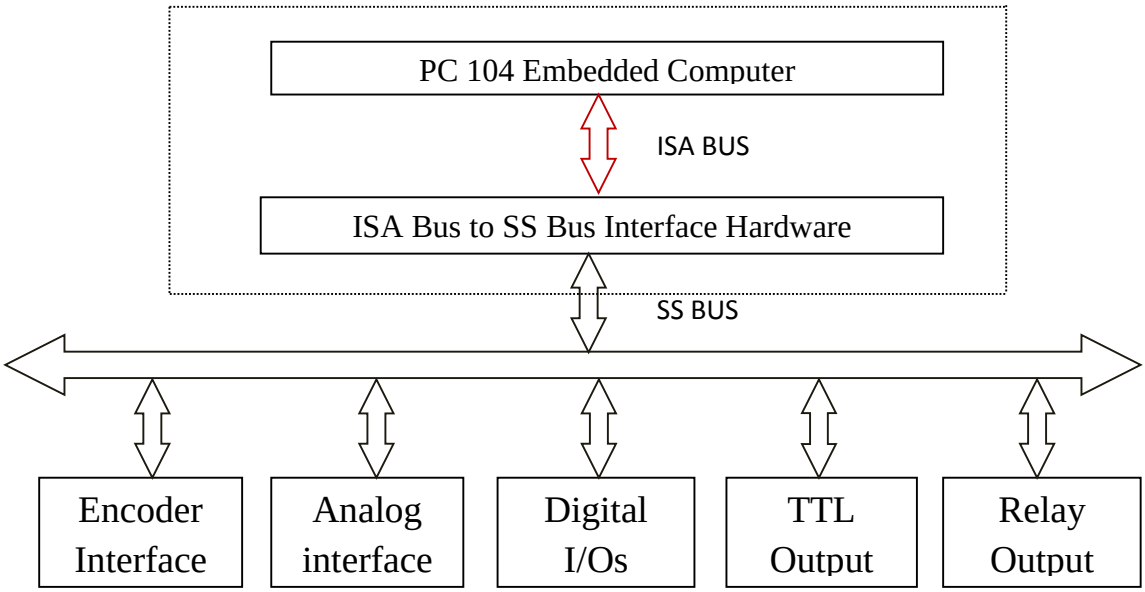


Figure 1-1 SSC Block Diagram

All control, data, and address signals which are currently available in the SSC bin are retailed well. For achieving bus compatibility, the bridge interface was designed between ISA and SS BUS. This module realizes a bridge logic which translates the ISA bus protocol to SS bus protocol and vice-versa. This bridge circuit implements address map registers and translation logic for mapping the ISA bus addresses to valid SS bus addresses.

1.1 Compatibility

The Table 1-1 shows the compatibility between PC104 SBC and existing 8086 SSC bin cards.

Slot No	***Existing SSC Bin***	***New SSC Bin***
1	CPU 8086	PC104
2	Real Time Clock	
3	Encoder card	Encoder Card
4	Discrete Input Card 1	Discrete Input Card 1
5	Discrete Input Card 2	Discrete Input Card 2
6	Analog I/O	Analog I/O
7	TTL Output	TTL Output
8	Relay Card	Relay Card

Table 1-1 Card Listing in SSC Bin

2. Description and Specification of SSBUS Signals

2.1 Description of SS BUS Signals

The GMRT SS BUS is parallel bus derived from the 8086 microprocessor memory and data buses along with control signals. This SS bus interacts with various resources of SSC Bin. (i.e. Encoder Card, Analog I/O Card, Digital Input Card, TTL & Relay Output cards.) The 8086 microprocessor resides in the CPU card of the SSC Bin and it becomes the default bus owner. All the SS Bus signals are extended to backplane through 64 pin euro connector. The maximum of 15 I/O boards can reside in the system bus.

2.2 SS BUS Signal Specification

- Data Bus Width = 16bit (D0 to D15)
- Address Bus Width = 20bit (A0 to A19)
- SS Bus Speed = 8MHz
- Data Transfer Mode = 8 bit Memory or I/O
- Total Memory Pages = 128 (7 bit page register)
- Paged memory access on SSBUS with 8KB memory page size (contiguous access)
- Paged I/O access on SSBUS with 32 bytes IO page size (HA0 & HA4 to HA7)

The SSBUS signals on the back plane of the station servo computer are listed below,

	A	C
1.	*GND*	*GND*
2.	*+V cc*	*+V cc*
3.	*SYS_CLK*	-
4.	*XACK_L*	RESET_L
5.	-	*HLDA_L*
6.	*IORD_L*	*IOWR_L*
7.	MRD_L	MWR_L
8.	INT_L	INTA_L
9.	*A0 (BLE)*	*BHE*
10.	*A1*	*A2*
11.	*A3*	*A4*
12.	*A5*	*A6*
13.	*A7*	*A8*
14.	*A9*	*A10*
15.	*A11*	*A12*
16.	*A13*	*A14*
17.	*A15*	A16
18.	A17	A18
19.	A19	LOCK_L
20.	*D0*	*D1*
21.	*D2*	*D3*
22.	*D4*	*D5*
23.	*D6*	*D7*
24.	D8	D9
25.	D10	D11
26.	D12	D13
27.	D14	D15
28.	INT1	INT2
29.	INT3	INT4
30.	*+15V*	*-15V*
31.	*GND*	*GND*
32.	*+V cc*	*+V cc*

Table 2-1 SSC Back Plane Signals Listing

- Power Supply & Ground:

V *cc supply of* +5 volts, +15 volts and GND are connected to the respective pins in the A & C rows of the 64 pin connectors.

- SSCLK:

This is the clock output from the CPU to the back plane of the SSBIN. In 8086 we are using the 24 MHz crystal to generate BCLK (Bus Clock) which is 1/3 of the input crystal frequency (i.e. 8 MHz) with 33% duty cycle.

- XACK_L:

This is the input signal to the CPU. This signal is asserted by the addressed off board devices after inserting required wait states,

Following addressed off board device is available in the SSC bin,

  - Discrete Input Board 1
  - Discrete Input Board 2
  - TTL Output Board
  - Mail Boxes in the AIO & Encoder Boards.

- RESET_L:

The RESET_L signal is derived from the Clock, Reset and Ready circuitry of 8086 CPU. This is used for resetting the off board devices during the Power On as well as Software reset.

- HLDA_L:

The HLDA_L goes HIGH during the DMA access cycles. The SS bus I/O cards deselects themselves during DMA cycles by gating the board select logic with HLDA_L signal.

- IORD_L / IOWR_L & MRD_L / MWR_L

These signals are control signals in the SS Bus for reading and writing the selected IO devices and memory address in the slave boards

1. Mail Boxes of Encoder & Analog IO Boards,
2. Discrete Input Board 1 & 2,
3. TTL Output Board

- INT_L & INTA_L:

INT_L signal is asserted by the Slaves boards to CPU. INTA_L is the output from the PIC in the CPU card as an acknowledge.

- Address Lines A0 to A19 & Data Lines D0 to D15:

These twenty address & sixteen data lines are available on the SSBUS to transfer data to the selected IO devices.

- INT1 to INT4:

These additional interrupt lines which are available in the SSBUS to get the interrupt from slave boards as well as to increase the number of interrupt inputs using slave board PICs with their INT output line connected to one of the bus interrupt lines.

Out of above mentioned 64 signals, the following signals are required to interface the PC104 SBC to the existing GMRT SS BUS.

SIGNALS	***TYPE****
A0 to A19	Output
D0 to D7	Bidirectional
IORD_L	Output
IOWR_L	Output
SYS_CLK	Output
XACK_L	Input
RESET_L	Output
HLDA_L	Output

Table 2-2 Required SSBUS signals for 8-bit I/O interfacing

* with respect to bus master (i.e. 8086 CPU SS bus)

2.3 Slave Board Port Address in GMRT Servo Computer

Encoder Board Address = 0x3xH

Encode Card Mail Box Read/Write = 0x30H

Encoder Card IMR Read/Write = 0x31H

Discrete Input Board 1 Address = 0x40H

Discrete Input Board 2 Address = 0x50H

Analog I/O Board Address = 0x6xH

Analog I/O Mail Box Read/Write = 0x60H

Analog I/O IMR Read/Write = 0x61H

TTL Output Board Address = 0x7xH

TTL Output Port 0 = 0x70H

TTL Output Port 1 = 0x71H

Summary of I/O Address map used in GMRT SSC Bin:

S.No	***Card Name***	***Access Type***	***Address Range***
1	Encoder Card	I/O Read/Write , BYTE	0x30H - 0x31H
2	Discreate Input Card 1	I/O Read/Write , BYTE	0x40H – 0x42H
3	Discreate Input Card 2	I/O Read/Write , BYTE	0x50H – 0x52H
4	Analog I/O Card	I/O Read/Write , BYTE	0x60H – 0x61H
5	TTL Output Card	I/O Read/Write , BYTE	0x70H – 0x71H

Table 2-3 Summary of SSBUS I/O address

3. Industry Standard Architecture

Industry Standard Architecture (ISA) is a computer bus standard for IBM PC compatible computers introduced with the IBM Personal Computer to support its Intel 8088 microprocessor's 8-bit external data bus and extended to 16 bits for the IBM Personal Computer/AT's Intel 80286 processor.

The PC/XT-bus is an eight-bit ISA bus used by Intel 8086 and Intel 8088 systems in the IBM PC and IBM PC XT in the 1980s. Among its 62 pins were de-multiplexed and electrically buffered versions of the eight data and 20 address lines of the 8088 processor, along with power lines, clocks, read/write strobes, interrupt lines, etc. The XT bus architecture uses a single Intel 8259 PIC, giving eight vectorized and prioritized interrupt lines. It has four DMA channels originally provided by the Intel 8237, three of the DMA channels are brought out to the XT bus expansion slots.

The PC/AT-bus, a 16-bit (or 80286-) version of the PC/XT bus, was introduced with the IBM PC/AT. This bus was officially termed *I/O Channel* by IBM. It extends the XT-bus by adding additional four address lines for total of 24 and eight data lines for a total of 16. It also adds new interrupt lines connected to a second 8259 PIC (connected to one of the lines of the first) and four 16-bit DMA channels, as well as control lines to select 8 or 16 bit transfers.

3.1 ISA Bus Specification

- ISA Bus Speed = 8MHz
- Data bus width = 16
- Address bus width = 24
- Transfer Rate = 64 Mbits/sec (Theoretical)

3.2 I/O Address Space

The maximum I/O address space supported by the ISA bus is 64KB (2^16). Only the first 10 address lines are decoded for I/O operations. This limits the I/O address space to address up to 0x3FFH. Some systems allow for 16 bit I/O address space, but may be limited due to some I/O cards only decoding 10 of these 16 bits.

The following list shows the reserved and available I/O address space in the 0x000H to 0x3FF range.

```
Port Address Assignments:
Port (hex)
000-00F   DMA Controller
010-01F   DMA Controller (PS/2)
020-02F   Master Programmable Interrupt Controller (PIC)
030-03F   Slave PIC
040-05F   Programmable Interval Timer (PIT)
060-06F   Keyboard Controller
070-071   Real Time Clock
080-083   DMA Page Registers
090-097   Programmable Option Select (PS/2)
0A0-0AF   PIC #2
0C0-0CF   DMAC #2
0E0-0EF   reserved
0F0-0FF   Math coprocessor, PCJr Disk Controller
100-10F   Programmable Option Select (PS/2)
110-16F   AVAILABLE
170-17F   Hard Drive 1 (AT)
180-1EF   AVAILABLE
1F0-1FF   Hard Drive 0 (AT)
```

```
200-20F   Game Adapter
210-217   Expansion Card Ports
220-26F   AVAILABLE
270-27F   Parallel Port 3
280-2A1   AVAILABLE
2A2-2A3   clock
2B0-2DF   EGA/Video
2E2-2E3   Data Acquisition Adapter (AT)
2E8-2EF   Serial Port COM4
2F0-2F7   Reserved
2F8-2FF   Serial Port COM2
300-31F   Prototype Adapter, Periscope Hardware Debugger
320-32F   AVAILABLE
330-33F   Reserved for XT/370
340-35F   AVAILABLE
360-36F   Network
370-377   Floppy Disk Controllers
378-37F   Parallel Port 2
380-38F   SDLC Adapter
390-39F   Cluster Adapter
3A0-3AF   reserved
3B0-3BB   Monochrome Adapter
3BC-3BF   Parallel Port 1
3C0-3CF   EGA/VGA
3D0-3DF   Color Graphics Adapter (CGA)
3E0-3EF   Serial Port COM3
3F0-3F7   Floppy Disk Controller
3F8-3FF   Serial Port COM1
```

3.3 Memory Address Space

The maximum memory address space supported by ISA bus is 16MB (2^24). The following list shows the system memory map.

```
00000-9FFFF System RAM (640k)
A0000-AFFFF EGA/VGA Video RAM
B0000-BFFFF Hercules/Mono/CGA Video RAM
C0000-C7FFF Video ROM
C8000-CFFFF Hard drive adapter BIOS ROM
D0000-D7FFF I/O Expansion ROM (unused on most systems)
D8000-DFFFF PC JR Cartridge (unused on most systems)
E0000-EFFFF Expansion ROM (unused on some systems)
F0000-FFFFF System ROM
100000+     System RAM (extended memory)
```

Available addresses

3.4 Signal Groups

The different signals in ISA bus are grouped into following groups

- Address and Data Group
- Cycle Control and Interrupt
- Direct Memory Access

The following tables list the various signals in the ISA bus.

8 BIT ISA			
Signal Name	***Pin***	***Pin***	***Signal Name***
Ground	B1	A1	I/O CH CK
AT RESET	B2	A2	ATD7
+5 V dc	B3	A3	ATD6
IRQ 9	B4	A4	ATD5
-5 V dc	B5	A5	ATD4
DRQ2	B6	A6	ATD3
-12 V dc	B7	A7	ATD2
NOWS	B8	A8	ATD1
+12 V dc	B9	A9	ATD0
Ground	B10	A10	ATI/O CH RDY
AT MEMW	B11	A11	ATAEN
AT MEMR	B12	A12	ATA19
AT IOW	B13	A13	ATA18
AT IOR	B14	A14	ATA17
DACK3	B15	A15	ATA16
DRQ3	B16	A16	ATA15
DACK1	B17	A17	ATA14
DRQ1	B18	A18	ATA13
REFRESH	B19	A19	ATA12
AT CLK	B20	A20	ATA11
IRQ7	B21	A21	ATA10
IRQ6	B22	A22	ATA9
IRQ5	B23	A23	ATA8
IRQ4	B24	A24	ATA7
IRQ3	B25	A25	ATA6
DACK2	B26	A26	ATA5
TC	B27	A27	ATA4
AT ALE	B28	A28	ATA3
+5 V dc	B29	A29	ATA2
OSC	B30	A30	ATA1
Ground	B31	A31	ATA0

Table 3-1 8-Bit ISA Signals

16 Bit ISA			
Signal Name	***Pin***	***Pin***	***Signal Name***
MEM CS16	D1	C1	SBHE
IO CS16	D2	C2	LA23
IRQ10	D3	C3	LA22
IRQ11	D4	C4	LA21
IRQ12	D5	C5	LA20
IRQ15	D6	C6	LA19
IRQ14	D7	C7	LA18
DACK0	D8	C8	LA17
DRQ0	D9	C9	MEMR
DACK5	D10	C10	MEMW
DRQ5	D11	C11	ATD08
DACK6	D12	C12	ATD09
DRQ6	D13	C13	ATD10
DACK7	D14	C14	ATD11
DRQ7	D15	C15	ATD12
+5 V dc	D16	C16	ATD13
MASTER	D17	C17	ATD14
Ground	D18	C18	ATD15

Table 3-2 16-Bit ISA Signals

The detailed description of various signal groups are given in appendix A.

Out of the above listed ISA BUS signals we are using only following signals for 8 bit I / O slave data transferring,

Signals	***Type****
ATA16 to ATA0	OUTPUT
ATD7 to ATD0	BIDIR
IOR	OUTPUT
IOW	OUTPUT
CLK	OUTPUT
RDY	INPUT
RESET	OUTPUT
AEN	INPUT

Table 3-3 Required ISA Bus signals for Interface

* with respect to ISA Bus

4. Address translation logic:

4.1 ISA BUS Addresses

The following table lists the valid range of memory and I/O addresses required to be issued on the ISA bus for accessing the bridge registers and SSBUS slaves.

Devices	***Address Range***	***Type***
IO Block (SS Bus I/O Devices)	0x300H to 0x31FH	I/O
IO Page Register R0 (LO_IOBASE_ADDR)	0x344H	I/O
IO Page Register R1 (HI_IOBASE_ADDR)	0x345H	I/O
Memory Page Register (MEMBASE_ADDR)	0x346H	I/O
Memory Block (SSBUS Memory)	0xE0000H to 0xE1FFFH	MEMORY

Table 4-1 Required ISA Bus Addresses

4.1 ISA to SS BUS Address Mapping

The following table lists the addresses emitted by the bridge on the SSBUS for full 1MB address range on the ISA BUS.

ISA Bus Address	***Cycle***	***SS Bus Address***	***Bus Cycle***	***Remarks***
00000H - 002FFH	I/O	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5
00000H - 002FFH	M	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5
00300H - 0031FH	I/O	0[IOPR]0H - 0[IOPR]1FH	I/O	ATA_4 to ATA_0 -> SSA_4 to SSA_0 [IOPR] -> SSA_{15} to SSA_5 0 -> SSA_{19} to SSA_{16}
00300H - 0031FH	M	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5
00320H - 0FFFFH	I/O	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5
00320H - 0FFFFH	M	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5
10000H - DFFFFH	M	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5
E0000H - E1FFFH	M	[MPR]000 - [MPR]1FFFH	MEMORY	ATA_{12} to ATA_0 -> SSA_{12} to SSA_0 [MPR] -> SSA_{19} to SSA_{13}
E2000H - FFFFFH	M	00000H - 0001FH	NONE	ATA_4 to ATA_0 -> SSA_4 to SSA_0 0 -> SSA_{19} to SSA_5

Table 4-2 Address Mapping

The following tables shows the total I/O and memory pages and their usage

IO Page Size	***32 Bytes***
Total IO Pages	2^{11} (IOPR = 11 Bit Register) = 2048
Usable IO Page Numbers	1 to 255 (Corresponds to IO Address Range 00020 - 01FFF)
Memory Page Size	***8 KB***
Total Memory Pages	2^{7} (MPR = 7 Bit Register) = 128
Usable Memory Page Numbers	1 to 277 (Corresponds to Memory Address Range 02000 - FFFFF)

Type	***Page Register***	***SS Bus Address Range***
I/O Space (Page Size = 32 Bytes)	0	00000 - 0001F
	1	00020 - 0003F
	2	00040 - 0005F
	3	00060 - 0007F
	4	00080 - 0009F
	,,	,,
	,,	,,
	255	01FE0 - 01FFF
Memory Space (Page Size =8 KB)	0	00000 - 01FFF
	1	02000 - 03FFF
	2	04000 - 05FFF
	3	06000 - 07FFF
	,,	
	7	0E000 - 0FFFF
	8 to 15	10000 - 1FFFF
	16 to 23	20000 - 2FFFF
	24 to 31	30000 - 3FFFF
	,,	,,
	,,	,,
	120 to 127	F0000 - FFFFF

Table 4-3 Summary of I/O and Memory Address Ranges

5. Generation of SS Bus Signals from PC104 ISA Bus Signals

- Data Lines (SS_D0 to SS_D7):

  The bidirectional data lines AT D0 to AT D7 are directly extended to the SS BUS.

- SSIORD_L & SSIOWR_L:

The two control signals are directly extended to the SSBUS through the SN54LS244. This Buffer is enabled by the SSEL_L signal. The SSSEL _L signal is generated in combination with the Bridge Enable, AT_IORD_L & AT_IOWR_L

- SSCLK:

This Clock is the same clock output of the ISA BUS (i.e. 8 MHz) which is directly connected to the SSBUS.

- SSRDY:

This signal is equivalent to XACK_L in the original 8086 CPU. This signal is input to PC104. XACK_L equivalent signal in the PC104 side is AT RDY.

- SSRESET_L:

SSRESET_L signal is derived from the AT RESET through the 54LS04.

- SHLDA_L:

This signal is always pulled HIGH in the bridge hardware. This signal is used in 8086 CPU to disable the salve boards during the DMA access.

- Address Lines(SA0 to SA19):

The generation of SS BUS Slave boards address from the ISA BUS address was given below.

Example:

Analog I/O Board address = xx6xH
Mail Box Read/Write address = 0060H
IMR Read/Write address = 0061H

The address blocks of 0x340-0x35FH (20th block of I/O Page) are used to send the address of the Slave Board and 0x300 to 0x31FH (18th block of I/O Page) are used to send the corresponding data to that specific salve board.

i.e outb (0x60, 0x344)
outb (0x00, 0x345)

the above two instructions carry the address of Analog I/O Mail Box. When 20th block address range will come on the address bus, CPLD address translation logic stores the data (i.e 0x60 & 0x00) in IO_BASE_REGISTER_LO & IO_BASE_REGISTER_HI.

outb(data, 0x300)

the above instruction send the data "data" in 18th block of I/O page (i.e. address range 0x300H to 0x031Fh). When this address will come on the address bus, CPLD address translation logic sends the translated address (i.e IO_BASE_REGISTER_LO & IO_BASE_REGISTER_HI) on the SS BUS insted of 0x300H and enables the IORD & IOWR signals.

The following block diagram gives the pictorial representation of interface hardware. The detailed schematics found in appendix B.

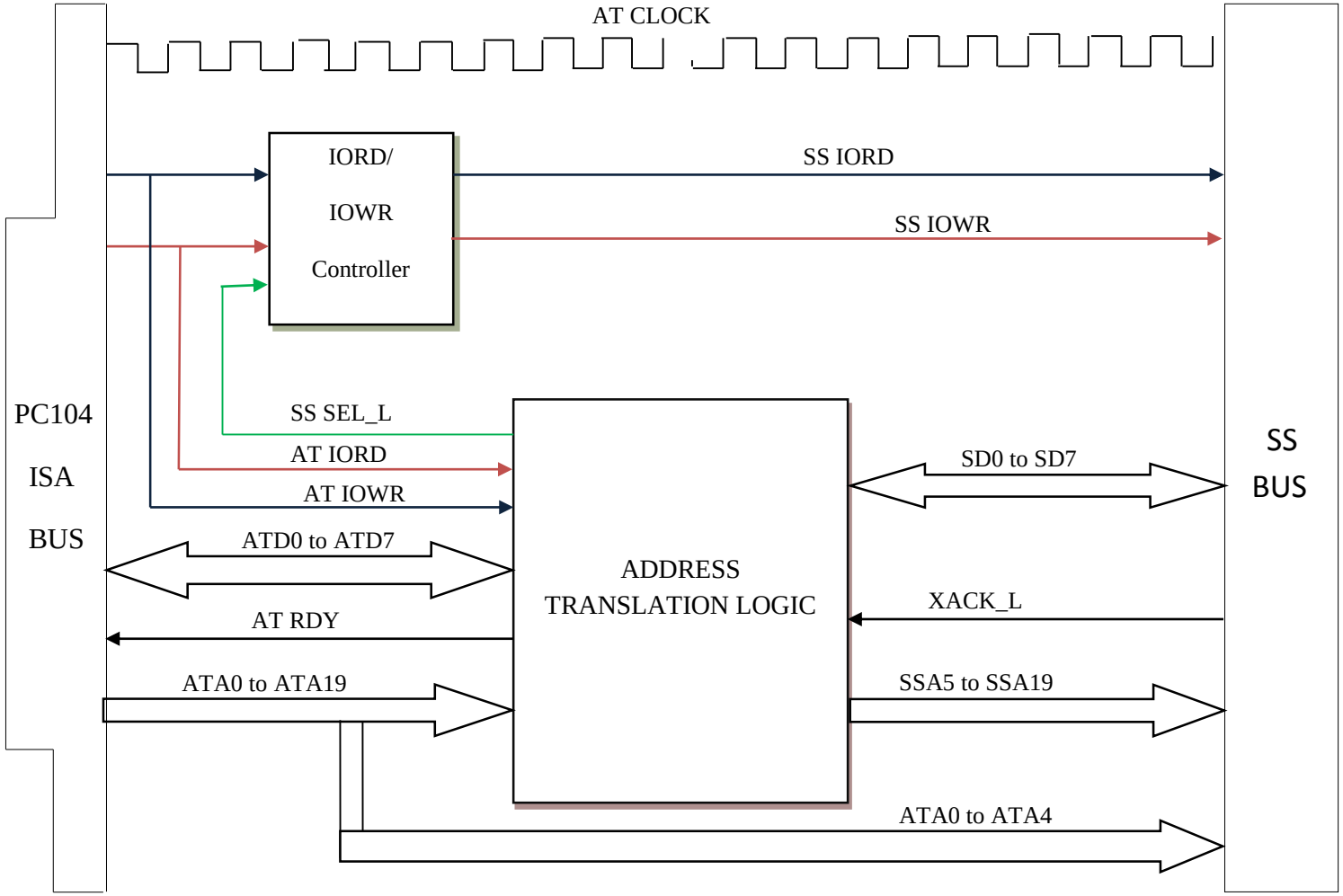


Figure 5-1 PC104 ISA Bus to SS Bus Hardware Interface Block Diagram

6. Circuit Description

The PC104 ISA bus to SS bus interface hardware translates the ISA address space into SS bus address space. The address translation logic is implemented in ALTERA EPM7128SLI84-10 cpld and circuits are assembled on a PCB which connects to ISA bus on one side and SS bus on other. A support board with 64 pin euro connector on one end and FRC connectors for extending signals from the bridge.

6.1 Mechanical Specification

- Size: 150 x 112 mm
- SS Bus Signals: 2 x 26pin FRC Connector
- AT Bus Signals: PC104 Connector

6.2 Technical Specification

The circuit can be divided into following functional groups

1. Address Decode & Control Signals
2. SS Bus Interface
3. Bus Termination

6.2.1 Address and Control Signal Decode

- The ALTERA EPM7128SLI84-10 does the address decoding for the SS bus addresses.
- It generates the SS bus RD/WR signals from the ISA bus RD/WR signals in conjunction with Bridge Enable signal for both memory and I/O operations.
- It generates the ATRDY signal from the XACK_L signal to terminate the I/O operation on addressed slave.

6.2.2 Bus Interface

The translated addresses from CPLD are sent to euro connector card through the buffer ICs U6 and U7 along with various control signals.

6.2.3 Bus Termination

Termination of the 8-bit and 16-bit ISA bus signals required to increase data integrity and system reliability. When termination is included, *AC termination* networks must be used to provide termination close to the characteristic impedance of the signal lines without exceeding the DC output current capabilities of the drivers.

As in the ISA standard, the recommended network consists of a resistor-capacitor network of 40-60 ohms in series with 30-70 pF, connected between each bus signal and ground. The termination network of 47 ohms in series with 33pF is connected in all address lines, Data Lines and control signals.

6.3 Jumpers and LED Specification

This interface hardware can be powered from either external power supply or backplane power supply by properly selecting the J3 jumpers.

External Power Supply - Close Right and Middle

Backplane Power Supply - Close Left and Middle

L1: SS Bus Read/Write Enable LED.

L2: AT Bus ALE Signal.

6.4 Connector Signal Details

FRC Connector P1

Signal Name	*Pin No*	*Pin No*	*Signal Name*
GND	1	2	GND
VCC BP	3	4	VCC BP
SS CLK	5	6	SSRESET_L
SSXACK_L	7	8	SHLDA_L
SSIORD_L	9	10	SSIOWR_L
NC	11	12	NC
ATA0	13	14	ATA1
ATA2	15	16	ATA3
ATA4	17	18	SSA5
SSA6	19	20	SSA7
SSA8	21	22	SSA9
VCC BP	23	24	VCC BP
GND	25	26	GND

Table 6-1 P1 FRC Connector Signals Details

FRC Connector P2

Signal Name	*Pin No*	*Pin No*	*Signal Name*
GND	1	2	GND
SSA10	3	4	SSA11
SSA12	5	6	SSA13
SSA14	7	8	SSA15
SSA16	9	10	SSA17
SSA18	11	12	SSA19
ATD0	13	14	ATD1
ATD2	15	16	ATD3
ATD4	17	18	ATD5
ATD6	19	20	ATD7
NC	21	22	NC
NC	23	24	NC
GND	25	26	GND

Table 6-2 P2 FRC Connector Signal Details

6.5 Component List

Resistors:

R1 ….. R32, R39, R40	=	47 ohms (SMD 0805 Package)
R44…. R60	=	47 ohms (SMD 0805 Package)
R33, R34, R41	=	4K7, 0.25W
R35	=	4K7 SIP Resistor
R36, R37, R38	=	1K, 0.25W
R42, R43	=	470E, 0.25W

Capacitors:

C1 …. C28, C30, C31, C32	=	33pF (SMD 0805 Package)
C42, C43, C48, C49 …C64	=	33pF (SMD 0805 Package)
C29, C46	=	10µF/25V
C33….. C41, C44, C45, C47, C65, C66	=	100nF (SMD 2012 Package)
C67….. C70	=	470µF/63V

Integrated Circuits:

U1, U6, U7	=	74HC244
U2	=	ALTERA EPM7128SLI84-10
U3	=	74HC04
U4	=	74HC32
U5	=	74LS125

Miscellaneous:

D3	=	1N4148
D1,D2	=	3 mm LED lamps
J1+J2	=	PC104 Connector
P1, P2	=	26 Pin FRC Connector

6.6 CPLD Program Listings

```
OPTIONS             BIT0 = LSB;

CONSTANT        IO_BLOCK            = H"018";
CONSTANT        MEM_BLOCK           = H"070";
CONSTANT        PIO_ADDR            = H"0320";
CONSTANT        LO_IOBASE_ADDR      = H"0321";
CONSTANT        HI_IOBASE_ADDR      = H"0322";
CONSTANT        MEMBASE_ADDR        = H"0323";
CONSTANT        NODE_ADDR_SW_ADDR   = H"0324";


SUBDESIGN bridge
(
     AT_RESET                           :INPUT;
     AT_CLK                             :INPUT;

     AT_MEMWR_L                         :INPUT;
     AT_MEMRD_L                         :INPUT;
     AT_IOWR_L                          :INPUT;
     AT_IORD_L                          :INPUT;

     AT_AEN                             :INPUT;
     AT_ADDR[19..0]                     :INPUT;
     TA[19..5]                          :OUTPUT;
     AT_DATA[7..0]                      :BIDIR;
     NODE_ADDR_SW[7..0]                 :INPUT;

     SS_XACK_L                          :INPUT;
     SS_RDY                             :OUTPUT;
     PIO_SEL_L                          :OUTPUT;
     SS_SEL_L                           :OUTPUT;
)


VARIABLE

     IO_BASE_REGISTER_LO[7..0]      :DFFE;
     IO_BASE_REGISTER_HI[7..0]      :DFFE;
     MEM_BASE_REGISTER[7..0]        :DFFE;
     DATA_BUS_BUFFER[7..0]          :TRI;
     BRIDGE_ENABLE                  :DFFE;




BEGIN
```

```
     DEFAULTS
          IO_BASE_REGISTER_LO[].ena     =    GND;
          IO_BASE_REGISTER_HI[].ena     =    GND;
          SS_SEL_L                      =    VCC;
          BRIDGE_ENABLE.ena             =    GND;
          DATA_BUS_BUFFER[7..0].oe      =    GND;
     END DEFAULTS;
     BRIDGE_ENABLE.clrn            =    !AT_RESET;
     IO_BASE_REGISTER_LO[].clrn    =    !AT_RESET;
     IO_BASE_REGISTER_HI[].clrn    =    !AT_RESET;
     MEM_BASE_REGISTER[].clrn =    !AT_RESET;

     BRIDGE_ENABLE.ena   =    ((AT_ADDR[15..0] ==
                                   NODE_ADDR_SW_ADDR) &
!AT_AEN);
     BRIDGE_ENABLE.d     =    AT_DATA[0];
     BRIDGE_ENABLE.CLK   =    AT_IOWR_L;

     IO_BASE_REGISTER_LO[].ena = ((AT_ADDR[15..0] ==
                                        LO_IOBASE_ADDR) &
!AT_AEN);
     IO_BASE_REGISTER_LO[7..0].d   =    AT_DATA[7..0];
     IO_BASE_REGISTER_LO[].clk     =    AT_IOWR_L;


     IO_BASE_REGISTER_HI[].ena = ((AT_ADDR[15..0] ==
                                        HI_IOBASE_ADDR) &
!AT_AEN);
     IO_BASE_REGISTER_HI[7..0].d   =    AT_DATA[7..0];
     IO_BASE_REGISTER_HI[].clk     =    AT_IOWR_L;


     MEM_BASE_REGISTER[].ena  = ((AT_ADDR[15..0] ==
                                        MEMBASE_ADDR) &
!AT_AEN);
     MEM_BASE_REGISTER[].clk       =    AT_IOWR_L;
     MEM_BASE_REGISTER[7..0].D     =    AT_DATA[7..0];


     PIO_SEL_L =    !(((AT_ADDR[15..0] == PIO_ADDR) #
(AT_ADDR[15..0] == PIO_ADDR+1) # (AT_ADDR[15..0] ==
PIO_ADDR+2) # (AT_ADDR[15..0] == PIO_ADDR+3)) & !AT_AEN);

     IF((AT_ADDR[19..5] == IO_BLOCK) & !AT_AEN &
RIDGE_ENABLE.q)
      THEN
           SS_SEL_L = !BRIDGE_ENABLE.q &
(!AT_IORD_L#AT_IOWR_L);
           TA[7..5]      =    IO_BASE_REGISTER_LO[7..5];
           TA[15..8]     =    IO_BASE_REGISTER_HI[7..0];
           TA[19..16]    =    GND;
```

```
     ELSIF ((AT_ADDR[19..13] == MEM_BLOCK) & !AT_AEN &
BRIDGE_ENABLE.q)
      THEN
           SS_SEL_L = !BRIDGE_ENABLE.q & (!AT_MEMRD_L #
AT_MEMWR_L);
           TA[19..13] = MEM_BASE_REGISTER[7..1];
           TA[12..5] = AT_ADDR[12..5];
     ELSE
           TA[19..5]     =    GND;
    END IF;

     SS_RDY    =    !SS_XACK_L;

     IF((AT_ADDR[15..0] == NODE_ADDR_SW_ADDR) & !AT_AEN)
     THEN
          DATA_BUS_BUFFER[].oe = !AT_IORD_L;
          DATA_BUS_BUFFER[7..0].in = NODE_ADDR_SW[7..0];
    END IF;
    AT_DATA[7..0] = DATA_BUS_BUFFER[7..0].out;

END;
```

Appendix A

Description of ISA Bus Signals

- ATA19 to ATA0

  System Address bits 19:0 are used to address memory and I/O devices within the system. These signals may be used along with LA23 to LA17 to address up to 16 megabytes of memory. Only the lower 16 bits are used during I/O operations to address up to 64K I/O locations. SA19 is the most significant bit. SA0 is the least significant bit. These signals are gated on the system bus when BALE is high and are latched on the falling edge of BALE. They remain valid throughout a read or write command. These signals are normally driven by the system microprocessor or DMA controller, but may also be driven by a bus master on an ISA board that takes ownership of the bus.

- LA23 to LA17

  Unlatched Address bits 23:17 are used to address memory within the system. They are used along with SA19 to SA0 to address up to 16 megabytes of memory. These signals are valid when BALE is high. They are "unlatched" and do not stay valid for the entire bus cycle. Decodes of these signals should be latched on the falling edge of BALE.

- AT AEN

  Address Enable is used to degate the system microprocessor and other devices from the bus during DMA transfers. When this signal is active the system DMA controller has control of the address, data, and read/write signals. This signal should be included as part of ISA board select decodes to prevent incorrect board selects during DMA cycles.

- AT ALE

  Buffered Address Latch Enable is used to latch the LA23 to LA17 signals or decodes of these signals. Addresses are latched on the falling edge of BALE. It is forced high during DMA cycles. When used with AEN, it indicates a valid microprocessor or DMA address.

- AT CLK

  System Clock is a free running clock typically in the 8MHz to 10MHz range, although its exact frequency is not guaranteed. It is used in some ISA board applications to allow synchronization with the system microprocessor.

- ATD15 toATD0

  System Data serves as the data bus bits for devices on the ISA bus. SD15 is the most significant bit. SD0 is the least significant bits. SD7 to SD0 are used for transfer of data with 8-bit devices. SD15 to SD0 are used for transfer of data with 16-bit devices. 16-bit devices transferring data with 8-bit devices shall convert the transfer into two 8-bit cycles using SD7 to SD0.

- DACK 0 to 3 & 5 to 7

  DMA Acknowledge 0 to 3 and 5 to 7 are used to acknowledge DMA requests on DRQ0 to DRQ3 and DRQ5 to DRQ7.

- DRQ 0 to3 & 5 to 7

  DMA Requests are used by ISA boards to request service from the system DMA controller or

to request ownership of the bus as a bus master device. These signals may be asserted asynchronously. The requesting device must hold the request signal active until the system board asserts the corresponding DACK signal.

- I/O CH CK

  I/O Channel Check signal may be activated by ISA boards to request than an non-mask able interrupt (NMI) be generated to the system microprocessor. It is driven active to indicate a uncorrectable error has been detected.

- AT I/O CH RDY

  I/O Channel Ready allow slower ISA boards to lengthen I/O or memory cycles by inserting wait states. This signals normal state is active high (ready). ISA boards drive the signal inactive low (not ready) to insert wait states. Devices using this signal to insert wait states should drive it low immediately after detecting a valid address decode and an active read or write command. The signal is release high when the device is ready to complete the cycle.

- AT IOR

  I/O Read is driven by the owner of the bus and instructs the selected I/O device to drive read data onto the data bus.

- AT IOW

  I/O Write is driven by the owner of the bus and instructs the selected I/O device to capture the write data on the data bus.

- IRQ 3 to 7, 9 to 12 & 14 to 15

  Interrupt Requests are used to signal the system microprocessor that an ISA board requires attention. An interrupt request is generated when an IRQ line is raised from low to high. The line must be held high until the microprocessor acknowledges the request through its interrupt service routine. These signals are prioritized with IRQ9 to IRQ12 and IRQ14 to IRQ15 having the highest priority (IRQ9 is the highest) and IRQ3 to IRQ 7 have the lowest priority (IRQ7 is the lowest).

- AT MEMR

  System Memory Read instructs a selected memory device to drive data onto the data bus. It is active only when the memory decodes is within the low 1 megabyte of memory space. SMEMR is derived from MEMR and a decode of the low 1 megabyte of memory.

- AT MEMW

  System Memory Write instructs a selected memory device to store the data currently on the data bus. It is active only when the memory decodes is within the low 1MB of memory space. SMEMW is derived from MEMW and a decode of the low 1MB of memory.

- MEMR

  Memory Read instructs a selected memory device to drive data onto the data bus. It is active on all memory read cycles.

- MEMW

Memory Write instructs a selected memory device to store the data currently on the data bus. It is active on all memory write cycles.

- REFRESH

  Memory Refresh is driven low to indicate a memory refresh operation is in progress.

- OSC

  Oscillator is a clock with a 70ns period (14.31818 MHz). This signal is not synchronous with the system clock (CLK).

- AT RESET

  Reset is driven high to reset or initialize system logic upon power up or subsequent system reset.

- TC

  Terminal Count provides a pulse to signal a terminal count has been reached on a DMA channel operation.

- MASTER

  Master is used by an ISA board along with a DRQ line to gain ownership of the ISA bus. Upon receiving a -DACK a device can pull -MASTER low which will allow it to control the system address, data, and control lines. After -MASTER is low, the device should wait one CLK period before driving the address and data lines, and two clock periods before issuing a read or write command.

- MEM CS16

  Memory Chip Select 16 is driven low by a memory slave device to indicate it is capable of performing a 16-bit memory data transfer. This signal is driven from a decode of the LA23 to LA17 address lines.

- I/O CS16

  I/O Chip Select 16 is driven low by a I/O slave device to indicate it is capable of performing a 16-bit I/O data transfer. This signal is driven from a decode of the SA15 to SA0 address lines.

- NOWS

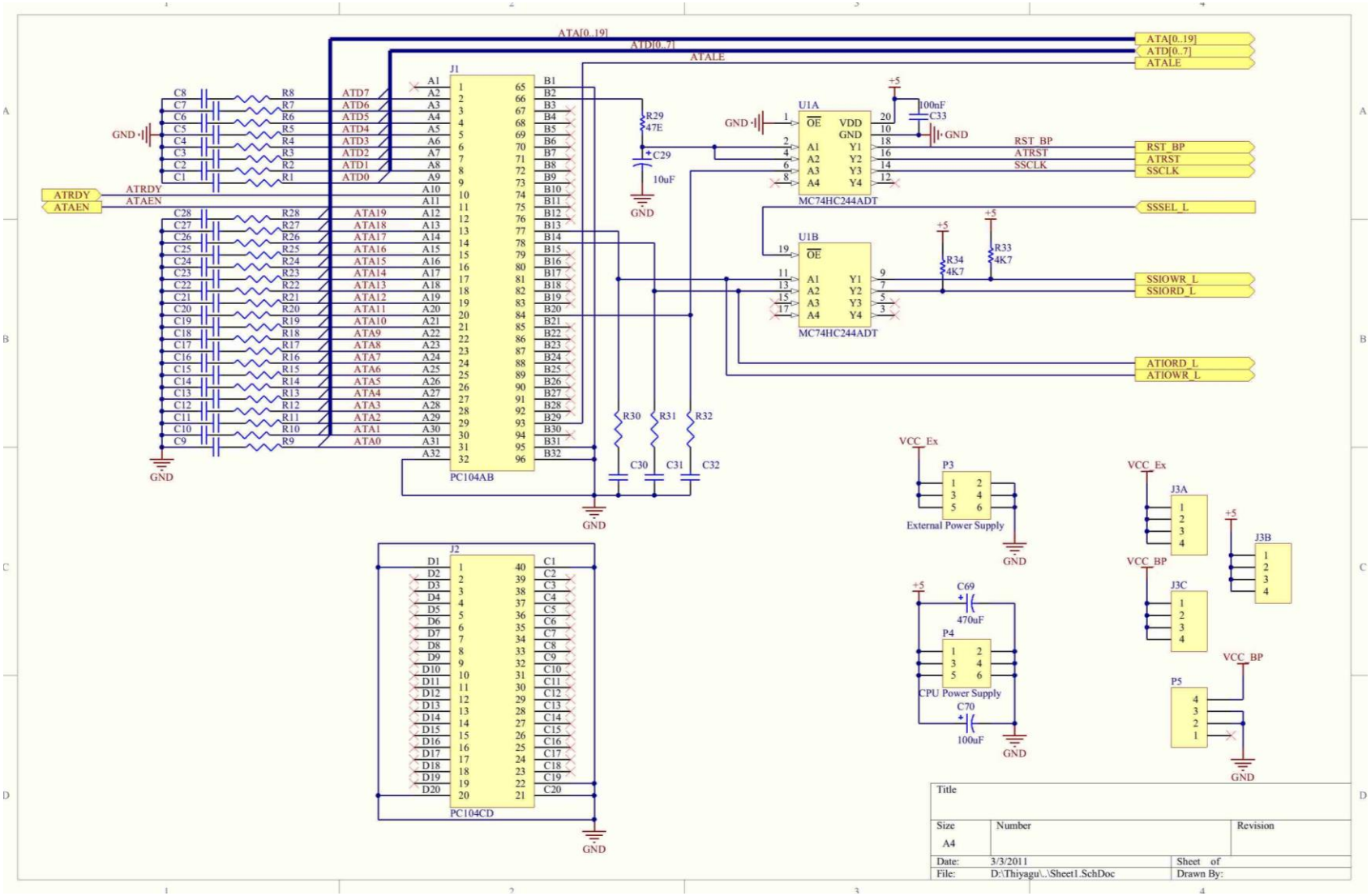
  Zero Wait State is driven low by a bus slave device to indicate it is capable of performing a bus cycle without inserting any additional wait states. To perform a 16-bit memory cycle without wait states, -0WS is derived from an address decode.

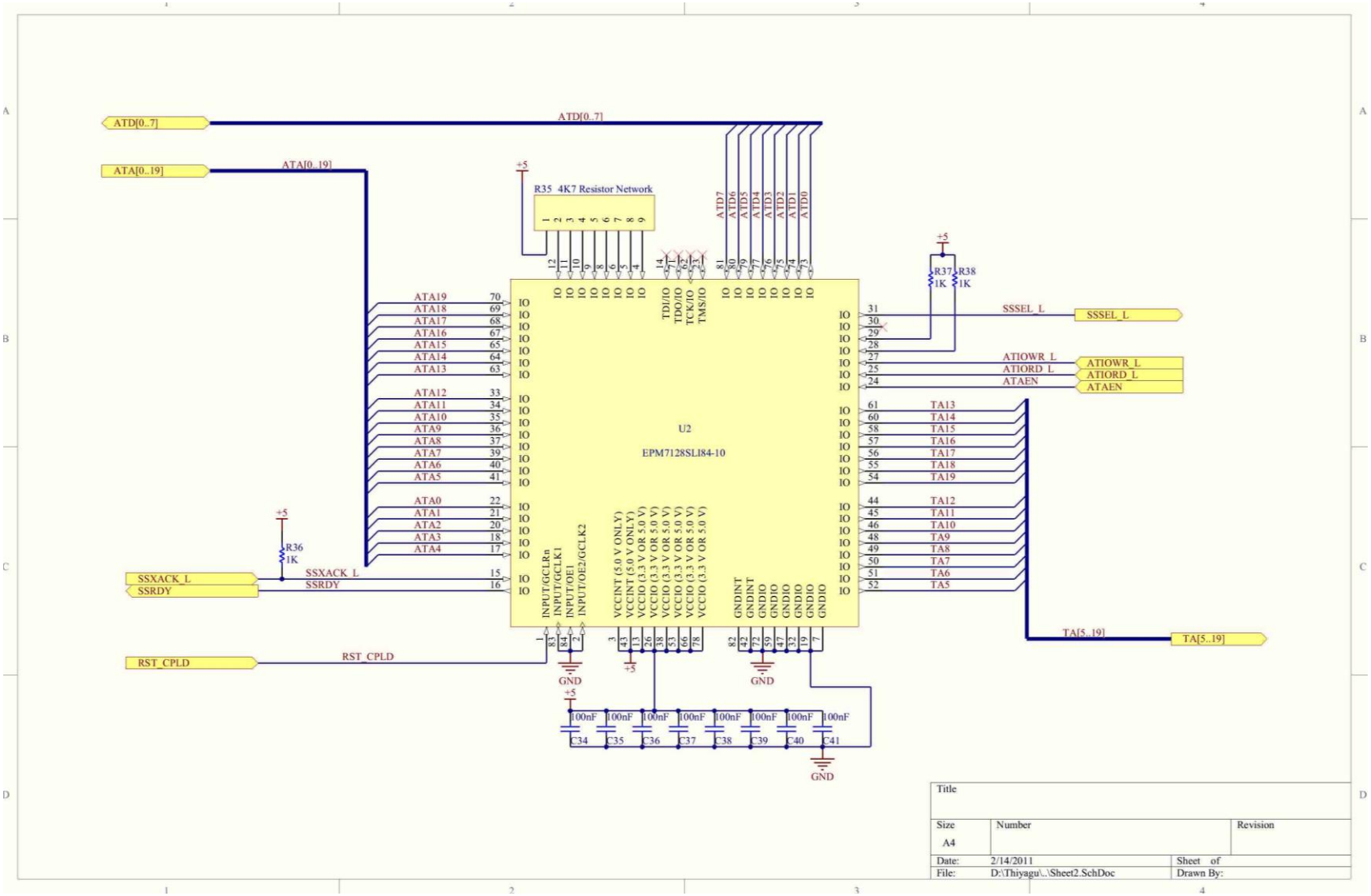
- SBHE

  System Byte High Enable is driven low to indicate a transfer of data on the high half of the data bus (D15 to D8).

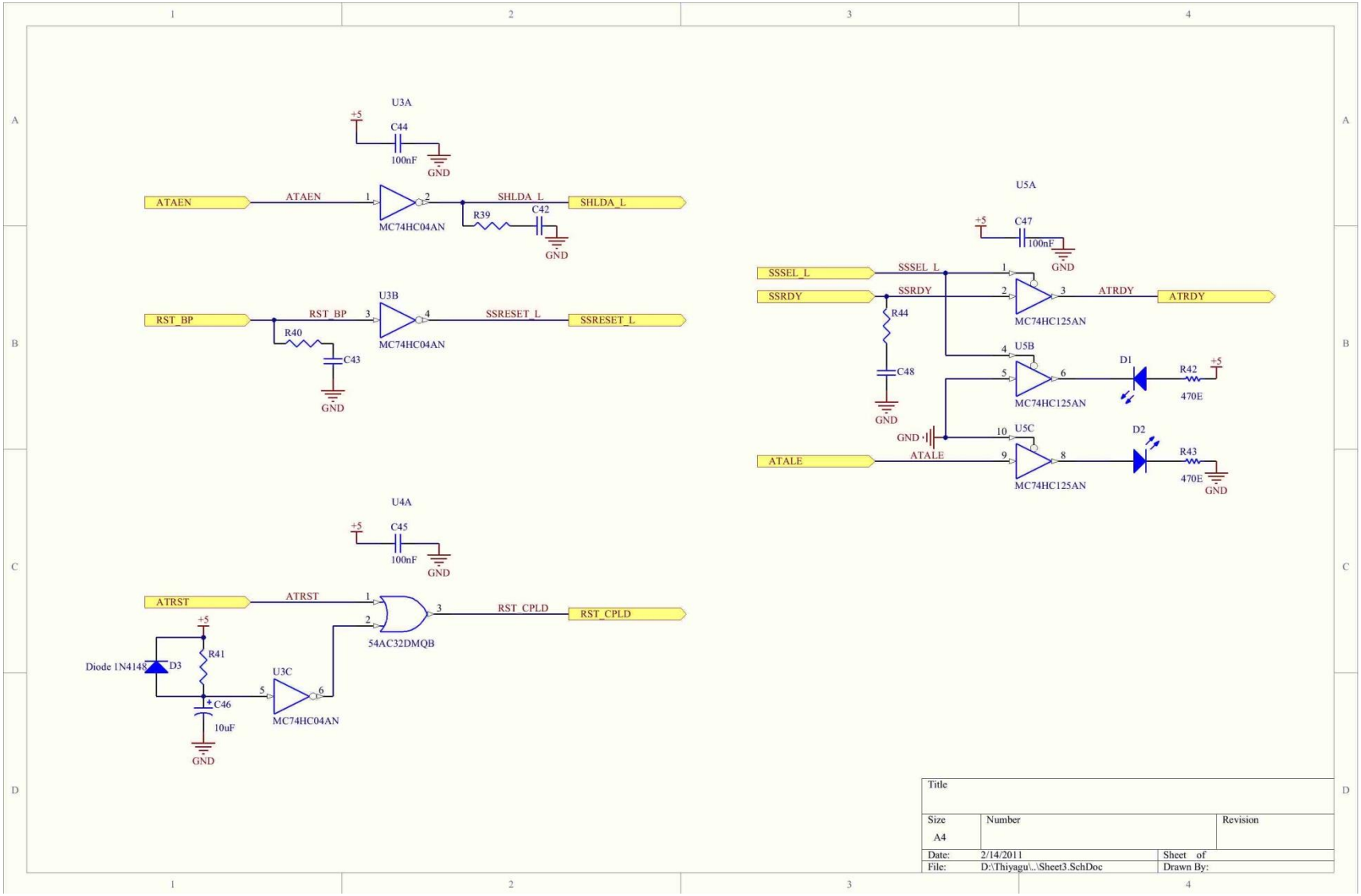
Appendix B

Interface Hardware Schematics

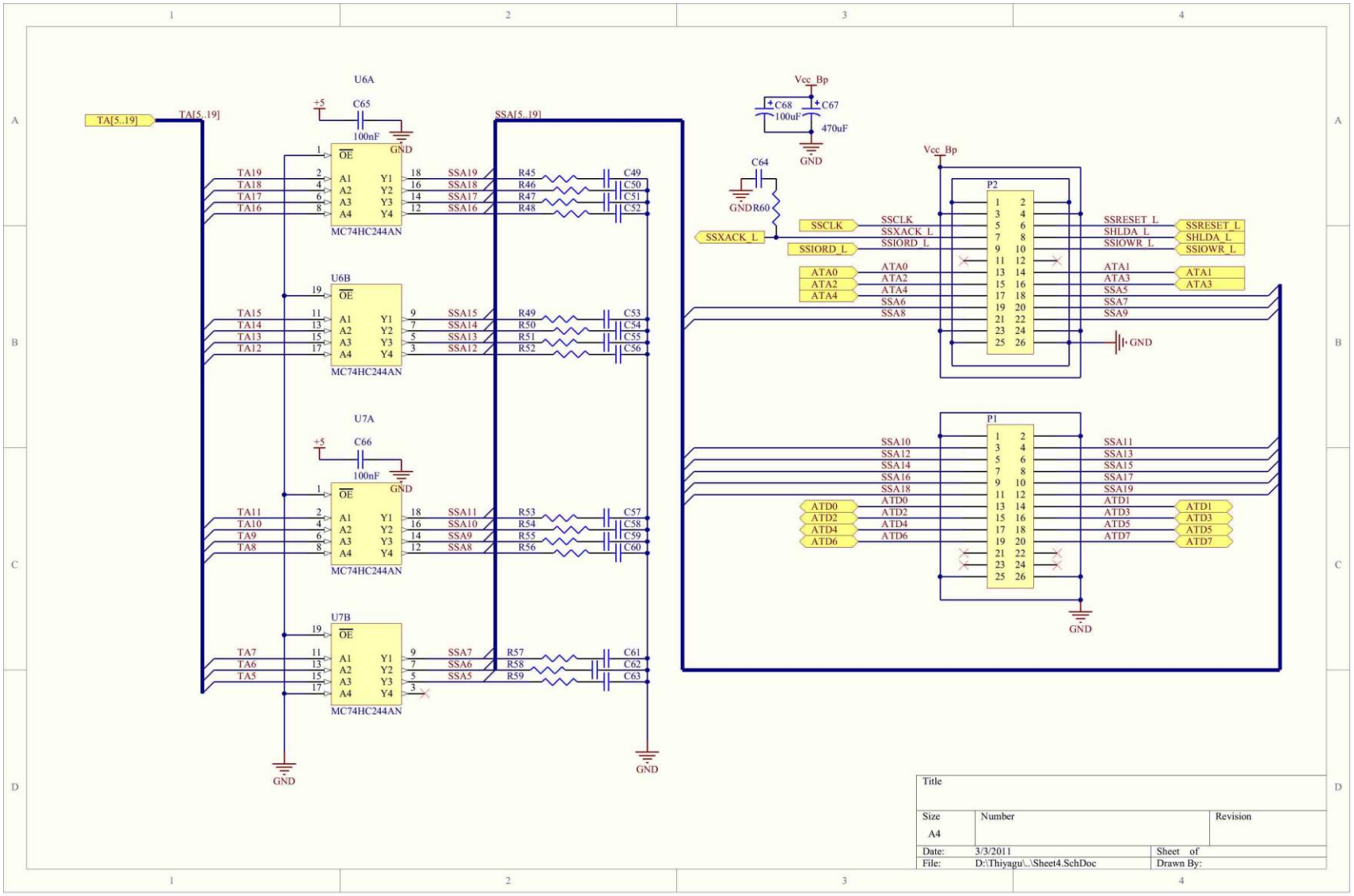

ATA[0..19]
ATD[0..7]
ATALE
J1
PC104AB
J2
PC104CD
ATRDY
ATAEN
GND
R29
47E
C29
10uF
U1A
U1B
MC74HC244ADT
100nF
C33
RST_BP
ATRST
SSCLK
SSSEL_L
SSIOWR_L
SSIORD_L
ATIORD_L
ATIOWR_L
R34
4K7
R33
4K7
R30
R31
R32
C30
C31
C32
VCC_Ex
P3
External Power Supply
C69
470uF
P4
CPU Power Supply
C70
100uF
VCC_BP
J3A
J3B
J3C
P5
Title
Size
A4
Number
Revision
Date:
3/3/2011
Sheet of
File:
D:\Thiyagu\..\Sheet1.SchDoc
Drawn By:

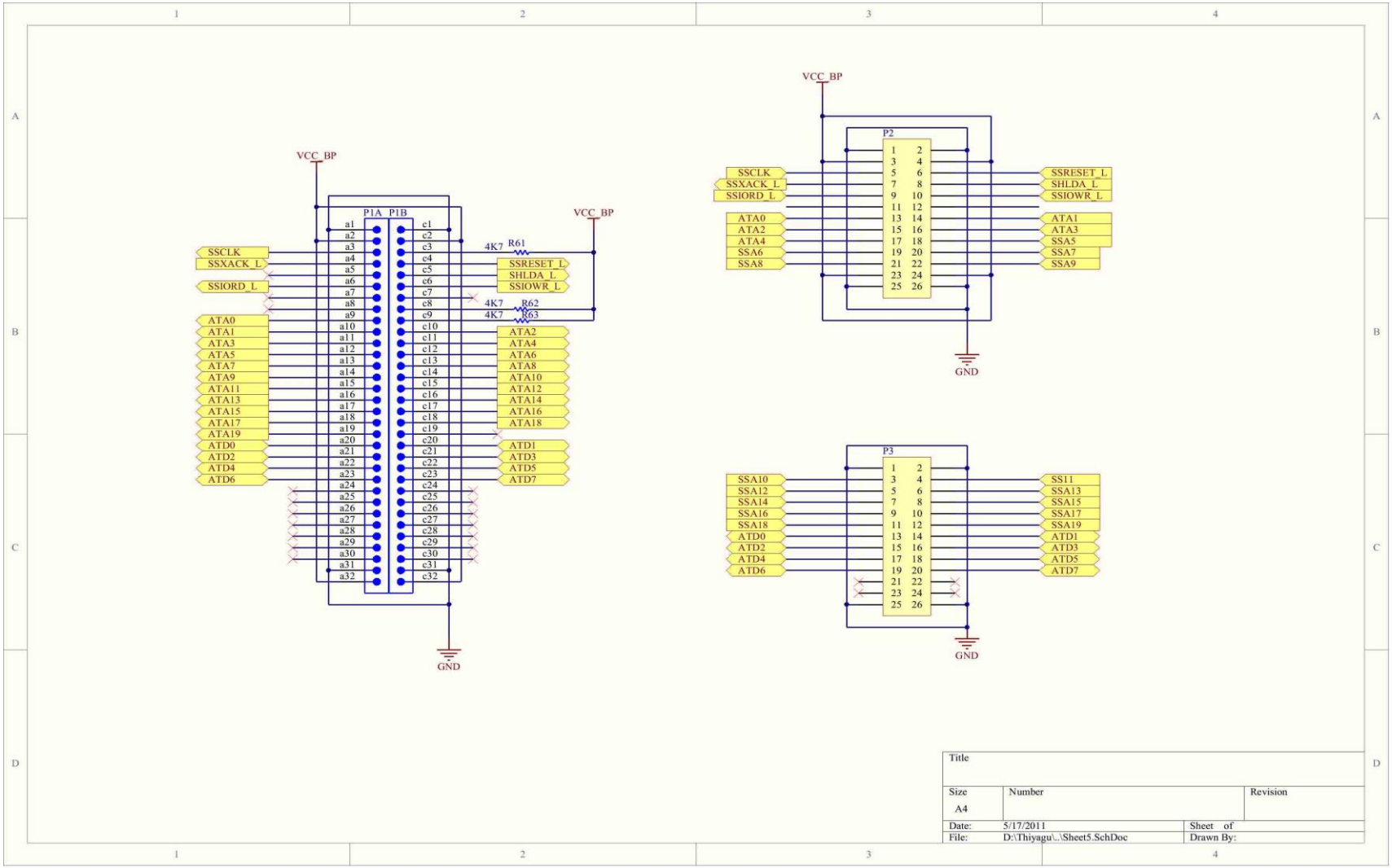


Title		
Size A4	Number	Revision
Date: 2/14/2011		Sheet of
File: D:\Thiyagu\..\Sheet2.SchDoc		Drawn By:



Title		
Size: A4	Number	Revision
Date: 2/14/2011		Sheet of
File: D:\Thiyagu\..\Sheet3.SchDoc		Drawn By:


U6A
U6B
U7A
U7B
MC74HC244AN
TA[5..19]
SSA[5..19]
C65
C66
100nF
GND
+5
TA19
TA18
TA17
TA16
TA15
TA14
TA13
TA12
TA11
TA10
TA9
TA8
TA7
TA6
TA5
SSA19
SSA18
SSA17
SSA16
SSA15
SSA14
SSA13
SSA12
SSA11
SSA10
SSA9
SSA8
SSA7
SSA6
SSA5
R45
R46
R47
R48
R49
R50
R51
R52
R53
R54
R55
R56
R57
R58
R59
C49
C50
C51
C52
C53
C54
C55
C56
C57
C58
C59
C60
C61
C62
C63
Vcc_Bp
C68
100uF
C67
470uF
C64
GNDR60
SSCLK
SSXACK_L
SSIORD_L
P2
P1
SSRESET_L
SHLDA_L
SSIOWR_L
ATA0
ATA2
ATA4
ATA1
ATA3
ATA5
ATA7
ATA9
SSA10
SSA12
SSA14
SSA16
SSA18
SSA11
SSA13
SSA15
SSA17
SSA19
ATD0
ATD2
ATD4
ATD6
ATD1
ATD3
ATD5
ATD7
Title
Size
A4
Number
Revision
Date:
3/3/2011
Sheet of
File:
D:\Thiyagu\..\Sheet4.SchDoc
Drawn By:


VCC_BP
P1A P1B
SSCLK
SSXACK_L
SSIORD_L
ATA0
ATA1
ATA3
ATA5
ATA7
ATA9
ATA11
ATA13
ATA15
ATA17
ATA19
ATD0
ATD2
ATD4
ATD6
4K7 R61
4K7 R62
4K7 R63
SSRESET_L
SHLDA_L
SSIOWR_L
ATA2
ATA4
ATA6
ATA8
ATA10
ATA12
ATA14
ATA16
ATA18
ATD1
ATD3
ATD5
ATD7
GND
P2
SSCLK
SSXACK_L
SSIORD_L
ATA0
ATA2
ATA4
SSA6
SSA8
SSRESET_L
SHLDA_L
SSIOWR_L
ATA1
ATA3
SSA5
SSA7
SSA9
P3
SSA10
SSA12
SSA14
SSA16
SSA18
ATD0
ATD2
ATD4
ATD6
SS11
SSA13
SSA15
SSA17
SSA19
ATD1
ATD3
ATD5
ATD7
Title
Size
A4
Number
Revision
Date: 5/17/2011
Sheet of
File: D:\Thiyagu\..\Sheet5.SchDoc
Drawn By:

Appendix C

Test Program and Timing Diagram

TTL Test Program:

---------- rtttl.c ----------

```
#include <rtai.h>
#include <rtai_sched.h>
#include <linux/module.h>
#include <linux/init.h>
#include <asm/io.h>
#define TICK_PERIOD 1000000000
#define TASK_PRIORITY 1
#define STACK_SIZE 10000

static RT_TASK ttl_task;

static void ttl(int t)
{
  static int i = 0;
  static unsigned char data;

  data =0xAA;

      outb(0xFF, 0x347);
      outb(0x60, 0x344);
      outb(0x00, 0x345);

  while(1)
   {
      outb(data, 0x310);
      outb(~data, 0x311);
      data = ~data;
      rt_busy_sleep(9000000);
      rt_task_wait_period();
   }
}

int init_module(void)
{
      RTIME tick_period;

      rt_set_periodic_mode();
      rt_task_init(&ttl_task, ttl, 1, STACK_SIZE, TASK_PRIORITY, 1, 0);
      tick_period = start_rt_timer(nano2count(TICK_PERIOD));
      rt_task_make_periodic(&ttl_task, rt_get_time() + tick_period,
tick_period);

      return 0;
}

void cleanup_module(void)
{
      stop_rt_timer();
      rt_task_delete(&ttl_task);
}
```

---------- Makefile ----------

```
KDIR  := /lib/modules/2.6.23/build

PWD         := $(shell pwd)

EXTRA_CFLAGS += $(shell /usr/realtime/bin/rtai-config –module-cflags)

export EXRTS_CFLAGS

all modules:
      $(MAKE)     -C $(KDIR) M=$(PWD) modules

modules_install:
      $(MAKE) –C $(KERNELDIR) M=$(PWD) modules_install

clean:
      $(MAKE) –C $(KDIR) M=$(PWD) clean

obj-m +=rtttl.o
```

Commands to run the program:

To build the kernel module issue “make command” as a root user

```
#make
```

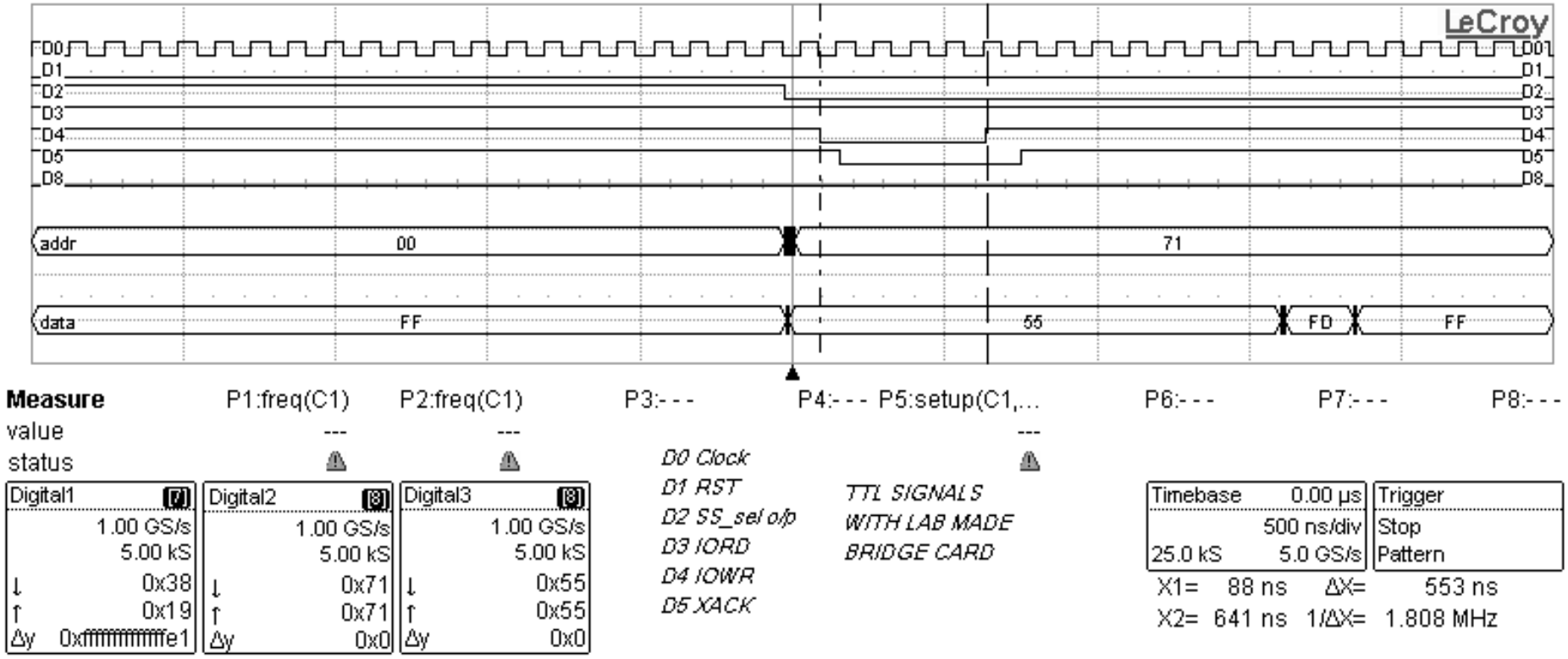
To drive the ttl port with 0xAAH insert the module using “insmod” command

```
#insmod rtttl.ko
```

To remove the module use “rmmod” module

```
#rmmod rtttl.ko
```

Timing Diagram:



Port Address	Data
0x70H	0xAA
0x71H	0x55

Digital Input:

---------- dip.c ----------

```
#include <rtai.h>
#include <rtai_sched.h>
#include <linux/module.h>
#include <linux/init.h>
#include <asm/io.h>

#define TICK_PERIOD 1000000000
#define TASK_PRIORITY 1
#define STACK_SIZE 10000

static RT_TASK dip_task;

static void dip(int t)
{
  static int p, n;
  static unsigned char data[6];


      outb(0xFF, 0x347);
      outb(0x40, 0x344);
      outb(0x00, 0x345);

  while(1)
   {
         for (n=0; n<2; n++)
         {
          for (p=0; p<3; p++)
          {
              data[(n*3)+p] = inb(0x300+(n*0x10)+p);
              //sleep(1);
          }
         }

       for(p=0; p<6; p++)
       {
          rt_printk("Port[%d] = %xH ",p,data[p]);
       }
      rt_printk("\n");
      rt_busy_sleep(9000000);
      rt_task_wait_period();
   }
}

int init_module(void)
{
      RTIME tick_period;

      rt_set_periodic_mode();
      rt_task_init(&dip_task, dip, 1, STACK_SIZE, TASK_PRIORITY, 1, 0);
      tick_period = start_rt_timer(nano2count(TICK_PERIOD));
      rt_task_make_periodic(&dip_task, rt_get_time() + tick_period,
tick_period);

      return 0;
}

void cleanup_module(void)
```

```
{
      stop_rt_timer();
      rt_task_delete(&dip_task);
}
```

---------- Makefile ----------

```
KDIR  := /lib/modules/2.6.23/build

PWD         := $(shell pwd)

EXTRA_CFLAGS += $(shell /usr/realtime/bin/rtai-config --module-cflags)

export EXRTS_CFLAGS

all modules:
      $(MAKE)     -C $(KDIR) M=$(PWD) modules

modules_install:
      $(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install

clean:
      $(MAKE) -C $(KDIR) M=$(PWD) clean

obj-m +=dip.o
```

Commands to run the program:

To build the kernel module issue “make command” as a root user

```
#make
```

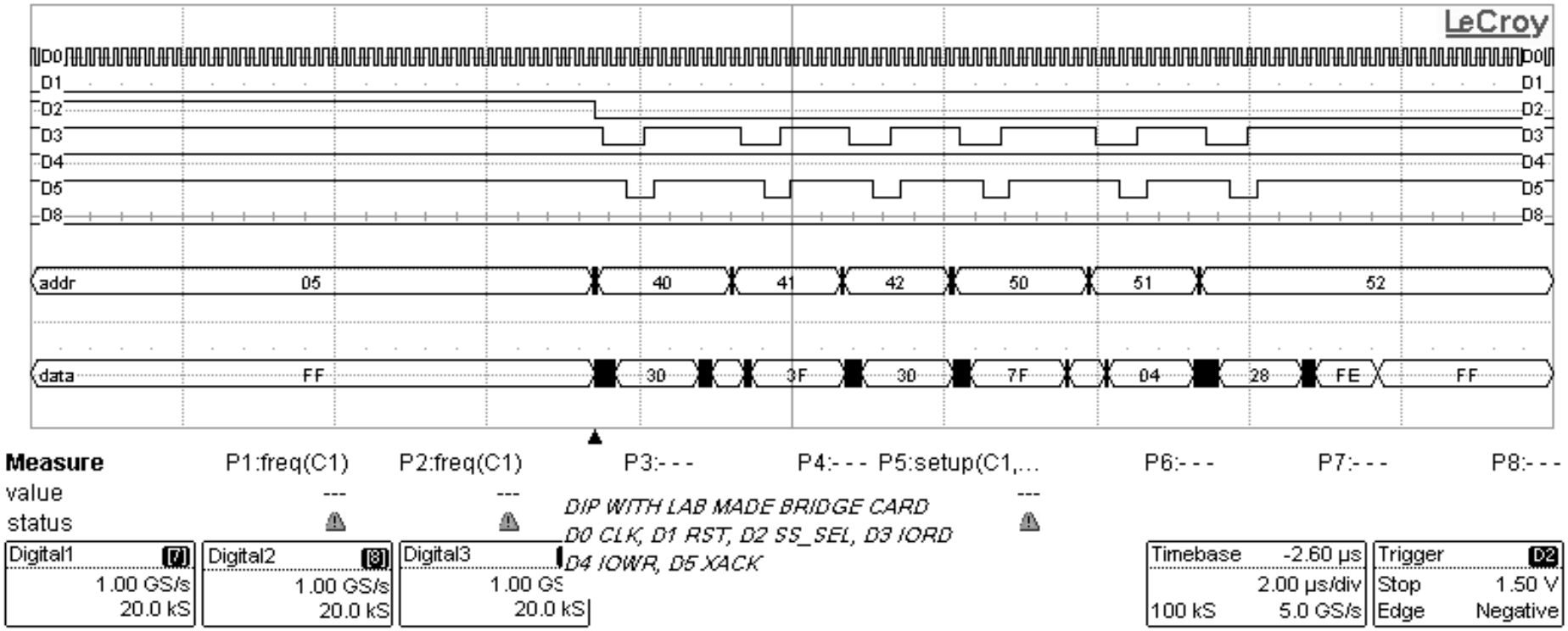
To drive the ttl port with 0xAAH insert the module using “insmod” command

```
#insmod dip.ko
```

To remove the module use “rmmod” module

```
#rmmod dip.ko
```

Timing Diagram:



Digital Input 1		***Digital Input 2***	
Port Address	**Data**	**Port Address**	**Data**
Port 0 (0x40)	0x30	Port 0 (0x50)	0x7F
Port 1 (0x41)	0x3F	Port 1 (0x51)	0x04
Port 2 (0x42)	0x30	Port 2 (0x52)	0x28

Encoder Card:

---------- encoder.c ----------

```
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_math.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
//#include <asm/io.h>
#include <float.h>


#define TICK_PERIOD 10000000
#define TASK_PRIORITY 0
#define STACK_SIZE 8192


#define BASEADDR       0x300
#define IO_BASEADDR_LO 0x344
#define IO_BASEADDR_HI 0x345
#define BRIDGE_ENABLE  0x347

#define HBF_H          0x01
#define LBF_H          0x02
#define ENC_MAILBOX    0x10
#define ENC_IMRSTAT    0x11
#define ENC_DATAPORT   BASEADDR + ENC_MAILBOX
#define ENC_STATPORT   BASEADDR + ENC_IMRSTAT

typedef struct ENC
{
 long RawBinval;
 float Enggval;
 float Softofst;
 unsigned char Winvel;
}ENC;

typedef struct
{
 unsigned char B0 : 8;
 unsigned char B1 : 8;
 unsigned char B2 : 8;
 unsigned char B3 : 8;
}LONG32BIT;

typedef struct
{
 unsigned char B0 : 1;
 unsigned char B1 : 8;
 unsigned char B2 : 8;
 unsigned char B3 : 8;
 unsigned char B4 : 7;
}LONG17BIT;



typedef union
{
```

```
 long L;
 LONG32BIT LB;
}FXDREAL;

typedef union
{
 long L;
 LONG17BIT LB;
}ENC17BIT;

static RT_TASK enc_task;
ENC encbrd[2];
float Enggval;

void Bridge_Enable(void)
{
 outb(0xFF, 0x347);
}

void Init_SlaveCards(void)
{
   unsigned char data;

   outb(0x20, 0x344);
   outb(0x00, 0x345);

 /* Clearing Mail Box Data */

   outb(0x02, ENC_STATPORT);
   data = inb(ENC_DATAPORT);
   rt_printk("Mail Box Data = %x\n",data);
}

int Enc_Reset(void)
{

  while(inb(ENC_STATPORT) & HBF_H);

  outb(0x0B, ENC_DATAPORT);

  while(!(inb(ENC_STATPORT) & LBF_H));
  if(inb(ENC_DATAPORT) == 0)
    return 0;
  else
    return 1;
}









void send_dispmode(void)
{
 int ix;
 unsigned char temp[2];
 char disp_mode;
```

```
 disp_mode = 'W';

 temp[0] = 0x02;

 switch(disp_mode)
   {
    case 'W':
      {
       temp[1] = 0x00;
       break;
      }

    case 'T':
      {
       temp[1] = 0x01;
       break;
      }

    case 'E':
      {
       temp[1] = 0x02;
       break;
      }

    case 'P':
      {
       temp[1] = 0x03;
       break;
      }

    case 'O':
      {
       temp[1] = 0xFF;
       break;
       }

    default:
       {
        temp[1] = 0x00;
        break;
       }
   }
 for(ix=0; ix<2;ix++)
 {
  while(inb(ENC_STATPORT) & HBF_H);
  outb(temp[ix], ENC_DATAPORT);
  }
}




void Read_EncEngg(ENC *enc)
{
 int ix;
 unsigned char temp[8];
 FXDREAL encdata;
 float enc_read;

 while(inb(ENC_STATPORT) & HBF_H);
```

```
 outb(0x08, ENC_DATAPORT);

 for(ix=0; ix<8; ix++)
  {
    while(!(inb(ENC_STATPORT) & LBF_H));
    temp[ix] = inb(ENC_DATAPORT);
    rt_printk("temp[%d] = %x\n",ix,temp[ix]);
  }
   /* Az Angle */

 encdata.L = 0;
 encdata.LB.B0 = temp[0];
 encdata.LB.B1 = temp[1];
 encdata.LB.B2 = temp[2];
 encdata.LB.B3 = temp[3];
// (enc)->Enggval = ((float)(encdata.L/0x10000));
 rt_printk("AZ Encoder Angle = %f\n",(float)encdata.L/0x10000);


 encdata.L = 0;
 encdata.LB.B0 = temp[4];
 encdata.LB.B1 = temp[5];
 encdata.LB.B2 = temp[6];
 encdata.LB.B3 = temp[7];
 //(enc+1)->Enggval = ((float)encdata.L/0x10000);
 rt_printk("EL Encoder Angle = %f\n",(float)encdata.L/0x10000);
}

void Read_EncBinary(void)
{
      int ix;
      unsigned char temp[6];
      ENC17BIT encdata;
      long Binval;

      while(inb(ENC_STATPORT) & HBF_H);
    outb(0x07, ENC_DATAPORT);

    for(ix=0; ix<6; ix++)
     {
       while(!(inb(ENC_STATPORT) & LBF_H));
       temp[ix] = inb(ENC_DATAPORT);
       rt_printk("temp[%d] = %x\n",ix,temp[ix]);
     }

      encdata.L = 0;
    encdata.LB.B0 = temp[0] & 0x01;
    encdata.LB.B1 = temp[1];
    encdata.LB.B2 = temp[2];
    encdata.LB.B3 = 0;
    encdata.LB.B4 = 0;
      rt_printk("AZ Binary = %ld\n",encdata.L);

      encdata.L = 0;
    encdata.LB.B0 = temp[3] & 0x01;
    encdata.LB.B1 = temp[4];
    encdata.LB.B2 = temp[5];
    encdata.LB.B3 = 0;
    encdata.LB.B4 = 0;
      rt_printk("EL Binary = %ld\n",encdata.L);
}
```

```
void data_acquisition(void)
{
 if (Enc_Reset())
   {
     rt_printk("Encoder Reset\n");
     send_dispmode();
   }
 Read_EncEngg(encbrd);
 Read_EncBinary();
}


static void *enc(int t)
{
  Bridge_Enable();
  Init_SlaveCards();
  while(1)
  {
   data_acquisition();
   rt_task_wait_period();
  }
 }

int init_module(void)
{

  RTIME tick_period;

  rt_set_periodic_mode();
  rt_task_init(&enc_task, enc,0, STACK_SIZE, TASK_PRIORITY, 0, 0);
  rt_task_use_fpu(&enc_task, 1);
  tick_period = start_rt_timer(nano2count(TICK_PERIOD));
  rt_task_make_periodic(&enc_task, rt_get_time() + tick_period,
tick_period);

  return 0;
}

void cleanup_module(void)
{
   stop_rt_timer();
   rt_task_delete(&enc_task);
}
```

---------- Makefile ----------

```
obj-m := encoder.o

KDIR  := /lib/modules/2.6.23/build /usr/lib

PWD         := $(shell pwd)

#EXTRA_CFLAGS := $(shell /usr/realtime/bin/rtai-config --module-cflags)
EXTRA_CFLAGS := -I/usr/realtime/include -I/usr/include -ffast-math -mhard-
float

#export EXRTS_CFLAGS
```

```
default:
	$(MAKE)	-C $(KDIR) M=$(PWD) modules

#modules_install:
#	$(MAKE) -C $(KERNELDIR) M=$(PWD) modules_install

#clean:
#	$(MAKE) -C $(KDIR) M=$(PWD) clean
clean:
	rm -f *.ko *.o .*.cmd .*.o.flags *.mod.c

#obj-m	+= encoder.o
```

Commands to run the program:

To build the kernel module issue “make command” as a root user

```
#make
```

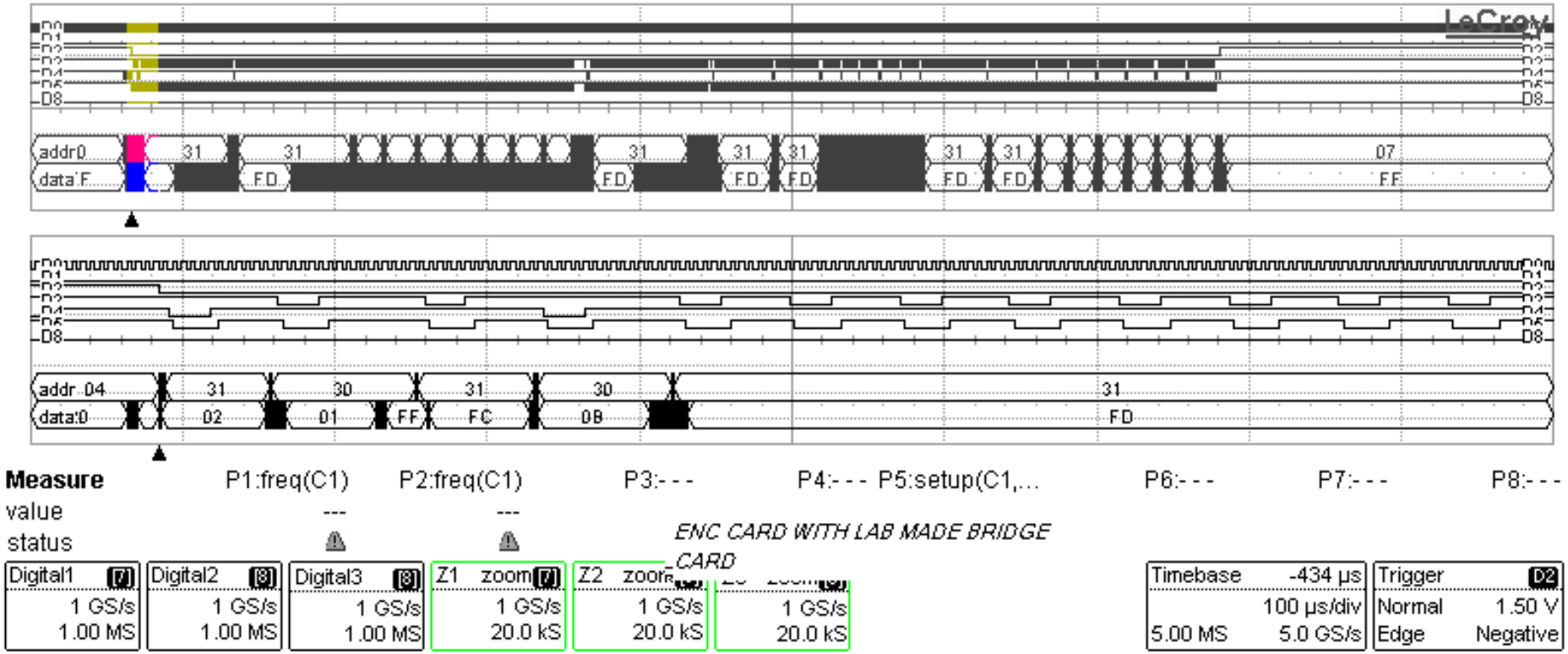
To drive the ttl port with 0xAAH insert the module using “insmod” command

```
#insmod encoder.ko
```

To remove the module use “rmmod” module

```
#rmmod encoder.ko
```

Timing Diagram



Port Address	Data
0x30H (Mail Box RD/WR)	0x01
0x31H (IMR)	0x02

Analog I/O Test Program

---------- analog.c ----------

```
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_math.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
//#include <asm/io.h>
#include <float.h>


#define TICK_PERIOD 100000000
#define TASK_PRIORITY 0
#define STACK_SIZE 8192


#define BASEADDR       0x300
#define IO_BASEADDR_LO 0x344
#define IO_BASEADDR_HI 0x345
#define BRIDGE_ENABLE  0x347

#define AIO_MAILBOX    0x00
#define AIO_IMRSTAT    0x01
#define AIO_DATAPORT   BASEADDR + AIO_MAILBOX
#define AIO_STATPORT   BASEADDR + AIO_IMRSTAT
#define HBF_H          0x01
#define LBF_H          0x02
#define ALLCHANNELS    12

RT_TASK resetcard;
RT_TASK read_ADC;
RT_TASK write_DAC;
RTIME period;

typedef short int WORD;
typedef unsigned char BYTE;
typedef int BOOL;

typedef struct
{
      float zs;
      float fs;
      float mult;
      BYTE hysterisis;
      float lmtli_val;
      float lmthi_val;
}ADC_CONFIG_TYPE;

typedef struct
{
      ADC_CONFIG_TYPE ConfigParam;
      WORD RawBinval;
      float Enggval;
      BOOL lmtlo_stat;
```

```
      BOOL lmthi_stat;
}ADC;

typedef struct
{
      WORD RawBinval;
      float Enggval;
}DAC;

typedef union
{
      long L;
      WORD W[2];
      BYTE B[4];
}Long;

typedef union
{
      WORD W;
      BYTE B[2];
}WordByte;

typedef struct
{
 unsigned char B0 : 8;
 unsigned char B1 : 8;
 unsigned char B2 : 8;
 unsigned char B3 : 8;
 }LONG32BIT;

typedef union
{
 long L;
 LONG32BIT LB;
}FXDREAL;

ADC adcbrd[12];
DAC dacbrd[2];

void Read_ADC_Engg(int chno)
{
      int ix,resp_len;
      BYTE temp[2], aiodata[64];
      FXDREAL val;
        char str[200];

      temp[0] = 0x02;
      temp[1] = chno;

      for(ix=0; ix<2; ix++)
      {
            while(inb(AIO_STATPORT) & HBF_H);
            outb(temp[ix], AIO_DATAPORT);
      }

      if(chno == ALLCHANNELS)
        resp_len = 48;
```

```
      else
        resp_len = 4;

      for(ix=0; ix<resp_len; ix++)
      {
            while(!(inb(AIO_STATPORT) & LBF_H));
            aiodata[ix] = inb(AIO_DATAPORT);
      }
      if(chno == 12)
      {
            for(ix=0; ix<chno; ix++)
          {  rt_printk("\n");
              val.LB.B0 = aiodata[(4*ix)];
              val.LB.B1 = aiodata[(4*ix)+1];
            val.LB.B2 = aiodata[(4*ix)+2];
              val.LB.B3 = aiodata[(4*ix)+3];

              rt_printk("a %d: %ld - %d
",ix,val.L,(int)((float)val.L/0x10000)*1000);
              sprintf(str, "%f", ((float)val.L/0x10000));
              rt_printk("A %s:",str);
          }
      }
      else
       {
              val.LB.B0 = aiodata[0];
              val.LB.B1 = aiodata[1];
            val.LB.B2 = aiodata[2];
              val.LB.B3 = aiodata[3];
            //  rt_printk("b %d: %d: ",ix, ((float)val.L/0x10000)*1000);
        }
        rt_printk("\n");
  }

void Read_ADC_Binary(int chno)
{
      int ix,resp_len;
      BYTE temp[2], aiodata[32];
      WordByte val;

      temp[0] = 0x01;
      temp[1] = chno;

      for(ix=0; ix<2; ix++)
      {
            while(inb(AIO_STATPORT) & HBF_H);
            outb(temp[ix], AIO_DATAPORT);
      }

      if(chno == 12)
        resp_len = 24;
      else
        resp_len = 2;
      rt_printk("\n");
      for(ix=0; ix<resp_len; ix++)
      {
            while(!(inb(AIO_STATPORT) & LBF_H));
```

```
            aiodata[ix] = inb(AIO_DATAPORT);
            //rt_printk("%x ",aiodata[ix]);
      }

      if(chno == 12)
      {
            for(ix=0; ix<chno; ix++)
          {
              val.B[0] = aiodata[(2*ix)];
              val.B[1] = aiodata[(2*ix)+1];
               // rt_printk("%04X ",val.W);
          }
      }
      else
       {
              val.B[0] = aiodata[0];
              val.B[1] = aiodata[1];
                //rt_printk("%04X ",val.W);
        }
        //rt_printk("\n");
  }


void Send_DAC_Engg(void)
{
      int ix;
      static int inc_cnt;
      BYTE temp[9];
      Long daceng;
      static float dacval;

      temp[0] = 0x04;

      dacval = 0.5;
      inc_cnt = 0;

      daceng.L = (long)(dacval*(inc_cnt++)*0x10000);
      rt_printk("%8d\n",daceng.L);
      temp[1] = daceng.B[0];
      temp[2] = daceng.B[1];
      temp[3] = daceng.B[2];
      temp[4] = daceng.B[3];

      daceng.L = (long)(dacval*(inc_cnt++)*0x10000);
      rt_printk("%8d\n",daceng.L);
      temp[5] = daceng.B[0];
      temp[6] = daceng.B[1];
      temp[7] = daceng.B[2];
      temp[8] = daceng.B[3];

      for(ix=0; ix<9; ix++)
      {
            while(inb(AIO_STATPORT) & HBF_H);
            outb(temp[ix], AIO_DATAPORT);
      }
}
```

```
void Write_anlg_Data(void)
{
      int ix;
      float aop[2];

      aop[0] = 0.1;
      aop[1] = 0.5;

      for(ix=0; ix<2; ix++)
      {
            dacbrd[ix].RawBinval = -(WORD)((aop[ix])*0x7fff);
            rt_printk("\n Dac[%d]: %x",ix,dacbrd[ix].RawBinval);
      }
}

void Send_DAC_Binary(DAC *dac)
{
      int ix;
      BYTE temp[5];
      WordByte dacval;

      temp[0] = 0x03;

      dacval.W = dac->RawBinval;
      temp[1] = dacval.B[0];
      temp[2] = dacval.B[1];

      dacval.W = (dac+1)->RawBinval;
      temp[3] = dacval.B[0];
      temp[4] = dacval.B[1];

      rt_printk("\nDAC BIN:");
      for(ix=0; ix<5; ix++)
      {
            while(inb(AIO_STATPORT) & HBF_H);
            outb(temp[ix], AIO_DATAPORT);
            rt_printk("%x ",temp[ix]);
      }
}

void *aio_thread(int j)
{
      while(1)
      {
//          Read_ADC_Binary(12);
//          Read_ADC_Engg(12);
        Write_anlg_Data();
        Send_DAC_Binary(dacbrd);
            rt_task_wait_period();
      }
}
```

```
void Send_ADC_Config(void)
{
      int ix;
      unsigned char temp[113];
      Long adczero,adcspan;

      temp[0] = 0x06;

      for(ix=0; ix<2; ix++)
      {
            temp[(8*ix)+1] = 0x00;
            temp[(8*ix)+2] = 0x00;
            temp[(8*ix)+3] = 0x00;
            temp[(8*ix)+4] = 0x00;

            temp[(8*ix)+5] = 0x00;
            temp[(8*ix)+6] = 0x00;
            temp[(8*ix)+7] = 0x00;
            temp[(8*ix)+8] = 0x00;
      }

    for(ix=2; ix<14; ix++)
    {
            adczero.L = (long)(0*0x10000);
            temp[(8*ix)+1] = adczero.B[0];
            temp[(8*ix)+2] = adczero.B[1];
            temp[(8*ix)+3] = adczero.B[2];
            temp[(8*ix)+4] = adczero.B[3];

            adcspan.L = (long)(100*0x10000);
            temp[(8*ix)+5] = adczero.B[0];
            temp[(8*ix)+6] = adczero.B[1];
            temp[(8*ix)+7] = adczero.B[2];
            temp[(8*ix)+8] = adczero.B[3];
      }

      for(ix=0; ix<113; ix++)
      {
            while(inb(AIO_STATPORT) & HBF_H);
            outb(temp[ix], AIO_DATAPORT);
      }
}

int AIO_Reset(void)
{
      int aio_cmd;

      aio_cmd = 0x05;
      while(inb(AIO_STATPORT) & HBF_H);
      outb(aio_cmd, AIO_DATAPORT);

      while(!(inb(AIO_STATPORT) & LBF_H));
      if(inb(AIO_DATAPORT) == 0)
        return 0;
      else
        return 1;
}
```

```
int init_module()
{
      outb(0xFF, BRIDGE_ENABLE);
      outb(0x60, IO_BASEADDR_LO);
      outb(0x00, IO_BASEADDR_HI);

      outb(0x02, AIO_STATPORT);
      rt_printk("Mail Box Data %x", inb(AIO_DATAPORT));

      if (AIO_Reset());
      {
            Send_ADC_Config();
      }

      rt_task_init(&read_ADC, aio_thread, 1, 8192, 0, 1, 0);
//    rt_task_init(&write_DAC, Write_DAC_Engg_thread, 0, 8192, 0, 1, 0);
      rt_task_use_fpu(&read_ADC, 1);
//    rt_task_use_fpu(&write_DAC, 1);
      rt_set_periodic_mode();

      period = start_rt_timer(nano2count(TICK_PERIOD));
//    period = nano2count(TICK_PERIOD);

      rt_task_make_periodic(&read_ADC, rt_get_time() + period, period);
//    rt_task_make_periodic(&write_DAC, rt_get_time() + period, period);

      return 0;
}


void cleanup_module()
{
      stop_rt_timer();
      outb(0x00, 0x347);
      rt_task_delete(&read_ADC);
//    rt_task_delete(&write_DAC);
}
```

---------- Makefile ----------

```
obj-m := analog.o

KDIR  := /lib/modules/2.6.23/build /usr/lib

PWD         := $(shell pwd)

EXTRA_CFLAGS := -I/usr/realtime/include -I/usr/include -ffast-math -mhard-
float

#export EXRTS_CFLAGS

default:
      $(MAKE)     -C $(KDIR) M=$(PWD) modules
```

```
#clean:
#	$(MAKE) -C $(KDIR) M=$(PWD) clean
clean:
	rm -f *.ko *.o .*.cmd .*.o.flags *.mod.c
```

Commands to run the program:

To build the kernel module issue “make command” as a root user

```
#make
```

To drive the ttl port with 0xAAH insert the module using “insmod” command

```
#insmod analog.ko
```

To remove the module use “rmmod” module

```
#rmmod analog.ko
```