

# ***Low voltage Modem Platform based on TMS320LC56***

Literature Number: BPRA049  
Texas Instruments Europe  
March 1997

## **IMPORTANT NOTICE**

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

**TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.**

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

---

## Contents

1. Introduction .....	1
2. DATA/PROG/MIPS analysis .....	2
3. Hardware design .....	4
3.1 System Overview .....	4
3.2 Connections to the Parallel Port.....	5
3.2.1 Memory Interface / Timing verification .....	7
3.2.2 Host Interface (UART) .....	11
3.3 Connecting the Analog Interface to the Buffered Serial Port .....	12
3.4 Variable System Clock .....	13
4. Enabling compatibility with TMS320LC541/3/6 .....	14
5. Software: Initialization & Test.....	15
5.1 Communication between DSP and PC via RS232.....	15
5.2 Buffered Serial Port and TLC320AC01 .....	16
5.3 Testing the external SRAM .....	17
Appendix A Source Code .....	19
Appendix B PAL Programming.....	35
Appendix C Schematics .....	39

## List of Figures

1 Figure 1: System Overview. ....	4
2 Figure 2: System Memory Map. ....	6

## List of Tables

1	Table 1: Status of memory select signals. ....	5
2	Table 2: Variable frequency for CLKIN. ....	13
3	Table 3: TMS320LC56 clock options. ....	13

## ***Low voltage Modem Platform based on TMS320LC56***

---

### **ABSTRACT**

This application report presents a modem reference design based on the TMS320LC56 DSP. The advanced peripherals of this device provide the features necessary for integration into complex systems optimizing access times and minimizing external logic. A software analysis at the beginning shows that the computing power of the LC56 is sufficient to support even the highest standards of voice and data transmission. The 3.3V technology of this DSP also makes it possible to integrate this solution in systems such as portable computers where low power consumption is the prime consideration. The combination of 5V with 3.3V devices is also discussed in this report. Onboard line interface, UART, large memory and expansion connectors bring a high degree of flexibility to this design, building a generic evaluation platform.

---

## **1. Introduction**

This application report discusses a modem reference design based on the TMS320LC56 DSP from Texas Instruments. A software analysis in the beginning will show that the TMS320LC56 has sufficient computing power to support even the highest standards of voice and data transmissions.

The flexible memory and I/O interfaces provide the features necessary to integrate the TMS320LC56 into complex systems with fast access times and minimum external control logic.

The 3.3V technology of the DSP allows power consumption to be minimized, making this solution appropriate for portable computers where power consumption is a prime consideration. Combining 5V and 3.3V devices will also be discussed in this report.

## 2. DATA/PROG/MIPS analysis

The modem industry is moving away from fixed chip-set designs to a more flexible and future-proof solution based on programmable DSP technology. The benefits of DSP technology are summarized below:

- **Reduced Cost**

A single TMS320C5X can replace a 2- or 3-chipset solution, reducing board size, chip count and component interfacing. Easy upgrades can be achieved by loading revised code into an external EPROM without the need to design and replace expensive components.

- **Differentiated Product**

The programmability of the DSP enables system designers to blend standard algorithms such as Data pumps with their own in-house algorithms which will differentiate their product from others in the market.

Next, the memory and processing power requirements for a complete V.34+ solution based on the C5X family is presented. An interpolation to the C54X family can be obtained assuming 25% increased efficiency with respect to the C5X figures given in the study.

**MEMORY/MIPS REQUIREMENT FOR COMPLETE V.34 SOLUTION  
( Based on C5X family)**

	<u>Program</u>	<u>Data</u>	<u>Mips</u>
<b>FAX</b>			
V.21	0.5Kw	0.1Kw	2.0
V.27ter	2.0Kw	0.4Kw	4.0
V.29	2.5Kw	0.5Kw	4.0
V.17	5.0Kw	0.8Kw	6.0
<b>MODEM</b>			
V.23	0.6Kw	0.1Kw	4.5
V.22/V.22bis	2.7Kw	0.3Kw	4.0
V.32/V.32bis	8.0Kw	6.0Kw	14.5
V.34	12.0Kw	10.0Kw	30.0
<b>COMPRESSION/ ERROR DETECTION</b>			
V.42 / V.42bis	6.0Kw	32.0Kw	6.0
<b>AT COMMANDS</b>			
Basic Comm. + Parser	5.5Kw	4.0Kw	(AT commands are contained in the call set-up, therefore, no impact on MIPS figure)
Extended AT	5.8Kw	0.5Kw	
<b>DSVD</b>			
G.729A	9.0Kw	2.0Kw	12.0
G.MUX	4.0Kw	1.0Kw	3.0
<b>TOTAL PROG</b>	<b>63.6Kw</b>	(Store in FLASH EPROM or partially masked on DSP ROM. Some extra memory should be added for Interrupt service routines, etc.)	

**WORST CASE SCENARIO**

	<u>Program</u>	<u>Data</u>	<u>Mips</u>
V.34 + V.42+ V.42bis + DSVD+ GMUX	<b>31Kw</b>	<b>45Kw</b>	<b>51</b>



## 3. Hardware design

### 3.1 System Overview

The system contains the components necessary for a modem. Data transfers and signal processing will be handled only by the DSP. The TLC320AC01 provides the A/D and D/A converters for the receive and transmit channels. Its analog input and output are connected to a telephone line interface which fulfills the US standard, enabling tests in a real environment.

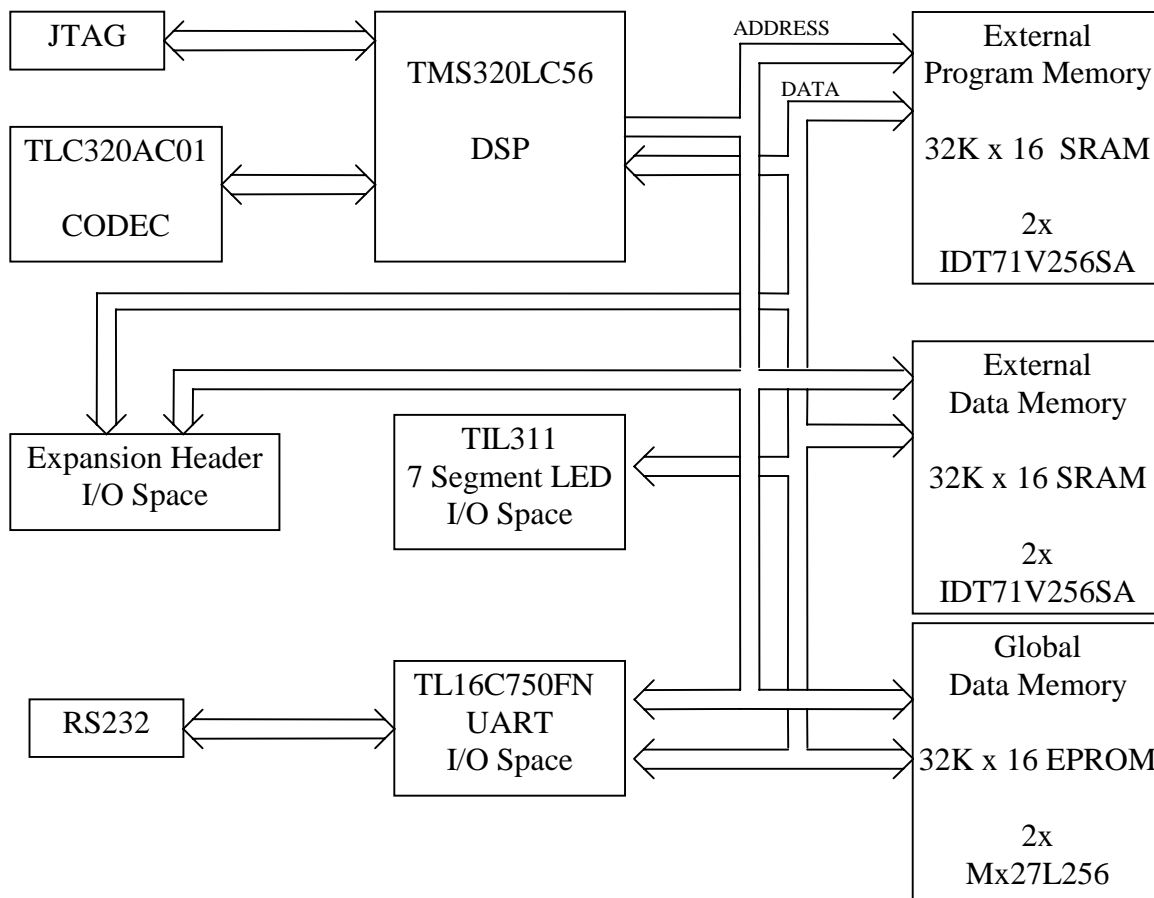
The JTAG port provides direct access to the DSP and optimizes test and emulation capabilities of the system. In addition, a seven segment LED display makes it possible to monitor the status of the DSP. It may be used for debug and demonstration purposes.

The UART TL16C750 enables the system to communicate with a host via an RS232 interface.

The large size of external memory provides a very flexible development environment.

An expansion connector gives the opportunity for additional peripherals.

Figure 1 shows the components of the system and the required connections of the bus signals.



**Figure 1: System Overview.**

### 3.2 Connections to the Parallel Port

The total space accessible from the DSP by the parallel port is divided into 'Program', 'Data', 'Global Data' and 'I/O'. With each access to external devices, the 'LC56 selects one of these spaces by driving the appropriate memory select strobes (PS\, DS\, IS\ and BR\) low:

**Table 1: Status of memory select signals.**

Enabled Space	PS\	DS\	IS\	BR\
Program	Low	High	High	High
Data	High	Low	High	High
Global Data	High	Low	High	Low
I/O	High	High	Low	High

The address and data lines are shared by all external spaces.

The two SRAM devices U5 and U6 (compare schematics) are the only memory devices in program space. The connection of PS\ to the CS\ inputs of the SRAMs is therefore sufficient to avoid any bus conflicts.

The EPROMs U7 and U8 must be placed in the global data memory for the boot load function stored in the on-chip ROM of the 'LC56. BR\ is connected to the chip enable of the EPROMs.

Because DS\ also goes low during global memory accesses it is not possible to use DS\ alone as chip enable for the SRAMs providing the external Data memory (U3 and U4). Bus conflicts would occur during each access to the EPROM.

One solution for avoiding this would be to generate the chip enable signal for the SRAMs with additional logic from BR\ and DS\. However, the additional delay is not acceptable when accessing the SRAMs with zero wait states in a 40MHz system.

To avoid any logic in the chip enable path it is better to use A15 as output enable of the SRAMs in addition to the connection of DS\ as chip select. Now the SRAM outputs are only enabled during accesses to the lower half (addresses 0x0000 - 0x7FFF) of the data space. EPROM is mapped in the global data space (addresses 0x8000 - 0xFFFF).

[User's Guide TMS320C5x, p. 6-40]

The I/O space will be shared by the 7 segment LED display, U18, and the UART communication element, U20. As both devices are not time critical it is possible to insert external logic to generate the chip enable signals as needed. To lengthen the memory cycles for these slower devices, software wait states have to be implemented for all accesses to the I/O space.

With all devices connected to the data bus, the capacitive loading is too high. The output drivers of the SRAMs are specified for external loads of only 30pF. The implementation of a buffer between the DSP and the slow devices like the UART, LED and the EPROM reduces the loading seen by the SRAMs.

	Program	Data	Global Data	I/O
0x0000		Memory- mapped Registers		
0x005F		On-Chip DARAM B2		
0x0060		Reserved		
0x007F				
0x0080		On-Chip DARAM B0 (CNF=0)		
0x00FF		Reserved (CNF=1)		
0x0100				
0x02FF		On-Chip DARAM B1		
0x0300		Reserved		
0x04FF				
0x0500	External SRAM U5 and U6 MP/MC=1	On-Chip SARAM Blk0		LED
0x07FF		BSP Block (OVLV=1)		
0x0800		External SRAM U3 and U4 (OVLV=0)		
0x0FFF				
0x1000		On-Chip SARAM Blk1		
0x17FF		BSP Block (OVLV=1)		
0x1800		External SRAM U3 and U4 (OVLV=0)		
0x1FFF				
0x2000		On-Chip SARAM Blk2		
0x3FFF		BSP Block (OVLV=1)		
0x4000	External SRAM U3 and U4 (OVLV=0)			
0x7FFF				
0x8000	On-Chip ROM MP/MC=0	External SRAM U3 and U4	External EPROM U7 and U8	Free
0xBFFF		Shadow external Data SRAM U3 and U4 (Write only)		UART
0xC000				External EPROM U7 and U8
0x97FF		On-Chip SARAM Blk0 (RAM=1)		
0x9800		Shadow of external SRAM U5 and U6 (RAM=0)		
0xFDFD		Shadow of external SRAM U5 and U6		
0xFE00		On-Chip DARAM B0 (CNF=1)		
0xFFFF		Shadow of external SRAM U5 and U6 (CNF=0)		

Figure 2: System Memory Map.

### 3.2.1 Memory Interface / Timing verification

#### a) SRAM

Read Cycle:

The SRAM devices used in this application are specified with an access time  $t_A = 15\text{ns}$  from address valid and CS\ low to data valid.

The 'LC56 starts any memory access cycle with the output of the valid addresses. At the same time the appropriate memory select strobe (DS\, PS\, BR\ or IS\ ) is driven low. After the time  $t_a(\text{RDAV})$  the data must be valid on the bus.

For address and chip select the following equation must be fulfilled:

$$t_A \leq t_a(\text{RDAV}) \quad (1)$$

In our case, with a 15ns SRAM, a TMS320LC56-80 and Clock frequency of 40MHz we have:

$$15\text{ns} \leq 15\text{ns} \quad \checkmark$$

( $\checkmark$  = equation fulfilled)

Before the DSP can latch the data with the rising edge of RD\ the bus has to be kept valid for the setup time  $t_{su}(\text{RD-RDH})$ . As RD\ is not connected to the SRAMs the timing is evaluated from the output of the valid address:

$$t_{su}(\text{RD-RDH}) \leq t_{wmin}(\text{RDH}) + t_{wmin}(\text{RDL}) - t_A; \quad (2)$$

$$7\text{ns} \leq H - 2\text{ns} + H - 2\text{ns} - 15\text{ns};$$

$$7\text{ns} \leq 6\text{ns}$$

In our case, equation (2) is not possible to fulfill. This worst case cannot occur as  $t_w(\text{RDH})$  and  $t_w(\text{RDL})$  are linked together. This means that when one of them has its minimum value the other one is usually longer for that time. The sum of the two will be much closer to 2H than  $2H - 4\text{ns}$  as calculated from equation (2) which ensures the setup time requirement,  $t_{su}(\text{RD-RDH})$ , is met.

After the DSP has latched the data with the rising edge of RD\, the bus will possibly no longer have kept valid ( $t_h(\text{RDH-RD}) = 0\text{ns}$ ).

## Write Cycle:

The falling edge on the WE\ input of the SRAM latches the addresses. The rising edge writes the data to the selected memory location.

The following equations must be fulfilled:

Address and CS\ valid before falling edge of WE\:

$$t_{AS} \quad t_{su}(AV-WEL) \quad (3)$$

$$0ns \quad 8.5ns \quad \checkmark$$

Minimum pulse duration WE\ low:

$$t_{WP} \quad t_w(WEL) \quad (4)$$

As OE\ of the SRAM is connected to ground permanently, the pulse duration of WE\ low must include the switching time of the SRAM's output drivers to high Z after the falling edge of WE\ and the setup time for the valid data before WE\ goes high again.

$$t_{WP} \quad t_{WHZ} + t_{DW}; \quad (5)$$

The combination of (4) and (5) gives the following:

$$t_{WHZ} + t_{DW} \quad t_w(WEL) \quad (6)$$

$$9ns + 7ns \quad 21ns \quad \checkmark$$

Before the rising edge of WE\ the data must be valid for the time  $t_{DW}$ :

$$t_{DW} \quad t_{su}(WDV-WEH) \quad (7)$$

$$7ns \quad 11ns \quad \checkmark$$

After the rising edge of WE\ the data has to be kept valid on the bus for the time  $t_{DH}$ :

$$t_{DH} \quad t_h(WEH-WDV) \quad (8)$$

$$0ns \quad 8.5ns \quad \checkmark$$

Before the write cycle can be completed with the rising edge of WE\ the addresses and the CE\ input have to be valid for the time  $t_{AW}$ :

$$t_{AW} \quad t_{su}(AV-WEL) + t_{wmin}(WEL); \quad (9)$$

$$10ns \quad 8.5ns + 21ns \quad \checkmark$$

**b) EPROM**

The EPROM devices used in this application are specified with an access time  $t_A = 150\text{ns}$  from address valid and CS\ low to data valid.

The 'LC56 starts any memory access cycle with the output of the valid addresses. At the same time the appropriate memory select strobe (DS\, PS\, BR\ or IS\) is driven low. After the time  $t_a(\text{RDAV})$  the data must be valid on the bus. The enable signal for the buffer (U22), driving the lower 8 bits of the data bus, is generated by a PAL device from IS\, DS\ and A15 using only one macro cell of the programmable device. During boot load 7 s/w wait states are added to each access to the global data memory. It can be assumed the buffer is enabled sufficiently before the EPROM to guarantee the output of valid data. Therefore, it is not necessary to take the buffer enable signal into account for timing verification.

For address and chip select the following equation must be fulfilled:

$$t_A \quad t_a(\text{RDAV}) + 7T - t_{\text{PHXmax}}(245); \quad (10)$$

$$150\text{ns} \quad 15\text{ns} + 175\text{ns} - 4.1\text{ns} \quad \checkmark$$

Before the DSP can latch the data with the rising edge of RD\ the bus has to be kept valid for the setup time  $t_{\text{su}}(\text{RD-RDH})$ . As RD\ is not connected to the EPROMs, the timing is evaluated from the output of the valid address:

$$t_{\text{su}}(\text{RD-RDH}) < t_{\text{wmin}}(\text{RDH}) + t_{\text{wmin}}(\text{RDL}) + 7T - t_A - t_{\text{PHXmax}}(245); \quad (11)$$

$$7\text{ns} < H - 2\text{ns} + H - 2\text{ns} + 175\text{ns} - 150\text{ns} - 4.1\text{ns}$$

$$7\text{ns} < 41.9\text{ns}; \quad \checkmark$$

To make the system compatible with the TMS320LC54x DSP the chip enable for the EPROM is created by a PAL using IS\, A15 and DS\. The additional delay for this gate causes no problems as the timing evaluations (10) and (11) are fulfilled with more than 35ns of tolerance.

**Note:**

The EPROM used in this application has an output disable time of 50ns. This may cause problems if an EPROM read cycle is immediately followed by a zero wait state access to another external device.

The use of the on-chip boot loader guarantees enough cycles between consecutive external accesses.

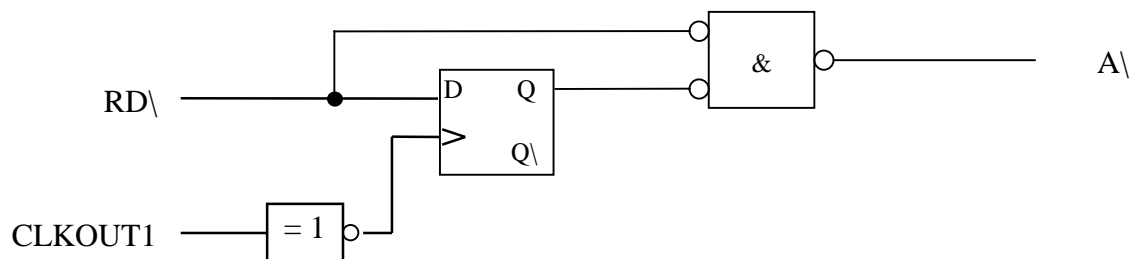
Definitions of names used in the equations above:

$t_a$ (RDAV)	Access time, LC56 reads data from address valid
$t_{su}$ (RD-RDH)	Setup time, data must be valid before RD high (LC56)
$t_h$ (RDH-RD)	minimum hold time, data must be valid after RD high (LC56)
$t_{wmin}$ (RDH)	minimum pulse duration of RD high (LC56)
$t_{wmin}$ (RDL)	minimum pulse duration of RD low (LC56)
$t_{su}$ (WDV-WEH)	setup time, LC56 outputs valid data before WE high
$t_{su}$ (AV-WEL)	setup time, LC56 outputs valid addresses before WE low
$t_{wmin}$ (WEL)	minimum pulse duration of WE low (LC56)
$t_h$ (WEH-WDV)	minimum hold time, LC56 keeps data valid after WE high
$t_A$	Memory access time, data output valid after control signals valid
$t_{WP}$	minimum pulse duration WE low required by SRAM
$t_{WHZ}$	SRAM's maximum delay time, output in high Z after WE low
$t_{DW}$	SRAM minimum setup time, data valid before WE high
$t_{DH}$	SRAM minimum hold time, data valid after WE high
$t_{AS}$	SRAM minimum setup time, address and CE valid before WE low
$t_{AW}$	SRAM minimum setup time, address and CE valid before WE high
$t_{PHXmax}$ (245)	Buffer maximum delay time, input to output valid
T	System clock cycle time
H	Half system clock cycle time T/2

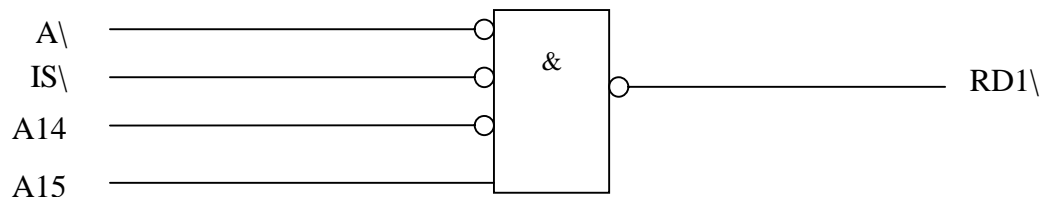
### 3.2.2 Host Interface (UART)

For the interface of the TL16C750 a compromise has been made. To keep the interface logic compatible with the TMS320LC54x DSPs and to avoid additional buffers in the address bus not all timing requirements of the specification for this device are met. Specifically, the read and write recovery times are not totally fulfilled. Therefore it may be necessary to implement 'NOP' cycles after any accesses to the UART in the software.

The following logic shifts the falling edge of RD\ for one clock cycle. This will ensure that the setup time for the addresses and CS\ is met. The NAND with RD\ ensures that the strobe goes high early enough to switch the output drivers in high impedance in time. This protects a subsequent access to another device from a bus conflict.

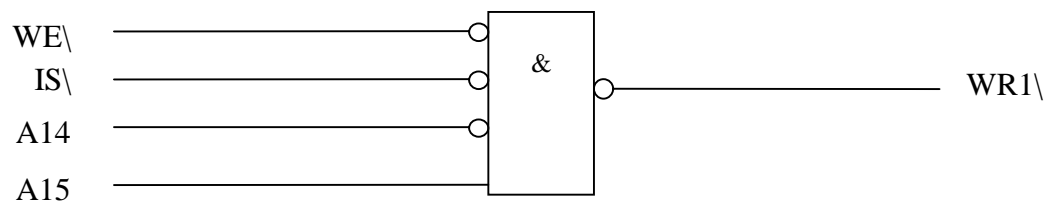


In addition the signal A\ has to be combined with the appropriate memory select signals to ensure that the output drivers are enabled only for accesses to the UART.



The two NAND gates in this logic can be merged and programmed as only one gate in the PAL.

It is not necessary to delay the falling edge of WE\ . It is just combined with the memory select signals:





### 3.3 Connecting the Analog Interface to the Buffered Serial Port

The TLC320AC01 provides the analog-to-digital and digital-to-analog conversion for the DSP. With the help of the headers J8 to J11 it is possible to switch the analog path between the line interface with connector J5 for modem applications and the phone connectors J12 (IN) and J13 (OUT) for other general purpose applications.

The TLC320AC01 derives the sampling frequency and the bandwidth of its internal low pass filter from the master clock. To enable a sampling rate of exactly 8kHz and the typical filter specification for telecom applications it is necessary to provide 10.368 MHz as MCLK frequency.

The low voltage DSP and the 'AC01 are connected to different power supplies. This may cause problems if one device drives the inputs of the other before it is connected to its power supply.

The connection of the voltage regulators, U13 and U14, in a chain prevents the AC01 from receiving any input signal from the DSP before its power supply voltage is applied.

Therefore it is only necessary to protect the DSP inputs from the 5V output voltage level of the AC01. A 5.6 k $\Omega$  series resistor (R7, R8, R9) limits the current into the DSP input to 1mA. An additional capacitor (C26, C27, C28) in parallel with this resistor maintains the rise and fall times. (for details refer to TLC320AC01 Application report, p.18, SLAAE09)

### 3.4 Variable System Clock

The clock of the DSP is generated by a programmable oscillator. The inputs A, B and C vary its output frequency on pin #2 as shown in table 2. Pin #1 always outputs the specified frequency of the device.

**Table 2: Variable frequency for CLKIN.**

Inputs			Outputs		
C	B	A		D	F
SW3	SW2	SW1		pin #2	pin #1
L	L	L	F/2	10MHz	20MHz
L	L	H	F/4	(5MHz)	20MHz
L	H	L	F/8	2.5MHz	20MHz
L	H	H	F/16	(1.25MHz)	20MHz
H	L	L	F/32	625kHz	20MHz
H	L	H	F/64	(312.5kHz)	20MHz
H	H	L	F/128	156.25kHz	20MHz
H	H	H	F/256	(78.125kHz)	20MHz

The pins DIV1 and DIV2 of the DSP specify the factor between the frequency on CLKIN and the DSP clock.

**Table 3: TMS320LC56 clock options.**

Parameter	CLKMD1 SW6	CLKMD2 SW5	CLKMD3 SW4
PLL multiply by three	L	L	L
PLL multiply by five	H	L	L
PLL multiply by four	L	H	L
PLL multiply by nine	H	H	L
External divide-by-two, oscillator disabled	L	L	H
PLL multiply by one	H	L	H
PLL multiply by two	L	H	H
Ext./Int. divide-by-two, oscillator enabled	H	H	H

## 4. Enabling compatibility with TMS320LC541/3/6

To be able to use the same design with a member of the TMS320LC54x family a second ZIF socket for a DSP has been added to the PCB. As both devices are connected in parallel to the bus it must be ensured that there is only one DSP inserted at a time.

As the parallel ports of the two processor types are slightly different, a couple of changes to the control logic are necessary.

The TMS320LC56's control outputs WE\ and RD\ of the parallel port do not exist on the TMS320LC54x devices. They have to be created by external logic for the UART and the SRAMs if the 'LC54x is used.

The TMS320LC56's STRB\ signal is split into the MSTRB and the IOSTRB strobes at the 'LC54x. MSTRB is only for accesses to the program and data space activated; IOSTRB only for accesses to the I/O space. The combination of R/W and MSTRB generates the WE\ strobe for writes to the SRAM banks. The RD1\ and WE1\ UART strobes are generated with IOSTRB, R/W\, and A[14:15] performing bank select.

To switch the board from 'LC56 to either 'LC541, 'LC543 or 'LC546 the PAL device, U19, has to be programmed with the file lead.jed instead of lc56.jed.

A jumper on J14 connects the WE\ output of the PAL to the SRAMs. This connection is only allowed if a DSP of the 'LC54x family is used. It must be removed for the 'LC56.

The 'LC56 must be placed into the socket U1, all 'LC54x devices into socket U24.

The boot loader of the LC54x reads the initialization word from the address 0xffff in the I/O space, while the boot load process itself accesses the EPROM in the data memory. Therefore the logic generating the enable signal for the EPROM must be combined in addition to DS\ and A15 with the IS\ strobe.

## 5. Software: Initialization & Test

The following chapter describes the programs for initialization and test routines written during the development of the board. The source code is contained in the appendix to this report.

### 5.1 Communication between DSP and PC via RS232

The Windows '*Terminal*' software enables the DSP to use the keyboard and monitor of a PC as input and display interface directly. The board transfers its data to the PC via the RS232 interface driven by the UART communication element, U20.

The data exchange between DSP and the communication element is controlled by the interrupts RXREADY and TXREADY driven by the UART. They are connected to the DSP as INT2 and INT3.

TXREADY signals to the processor that the previous byte has been sent to the PC and it is then possible to write the next byte to its transmit buffer. RXREADY tells the DSP a new byte has been received from the PC and is ready for reading.

To avoid any loss of data, the DSP must read a new byte from the UART receive channel before the PC sends the next byte. Therefore RXREADY is connected to an interrupt with a higher priority than TXREADY.

The software described in the following is based on the communication between PC and DSP via the UART.

The user can choose a routine to run from the menu. By pressing any key, the routine activated will end and the DSP will return to the main program, giving the user a new choice.

## 5.2 Buffered Serial Port and TLC320AC01

The analog interface TLC320AC01 is connected to the DSP via the buffered serial port. It consists of two independent channels for transmitting and receiving. Each channel has its own memory space or buffer. From the buffer, the transmit channel will read data to be sent. After each read the transmit buffer address pointer AXR will be incremented. The size of the transmit buffer is specified in the register BKX. The port will generate the XINT interrupt each time one half of the buffer has been transmitted.

If AXR has reached the end of the buffer it will be reset to the start address.

The receive channel is similarly organized. ARR contains the actual receive buffer address and BKR the length. The RINT interrupt is generated each time one half of the buffer has been loaded with new data.

The DSP can access the buffers as memory mapped into the On-Chip SARAM B1k0 starting at address 0x0800 in the data space.

The subroutine `_ac01` of the program will reset the 'AC01 via the XF pin and initializes the control registers of the analog interface. The DSP will then loop back the incoming samples to the output channel of the AC01.

The AC01 has primary communication for the data exchange from the A/D and D/A converters, and secondary communication to set its control registers. Secondary communication is activated if the two LSBs of a data word written to the DAC during primary communication are set high.

The initialization words used by the software are pre-loaded into the transmit buffer, and are sent interleaved with dummy data to activate secondary communication.

After initialization, it must be ensured that only data words are transmitted to the AC01 with bits 0 and 1 set to zero, unless secondary communication is requested to change the setup of the analog interface. The program then starts to receive samples and store them in the transmit buffer before any samples are sent to the AC01.

After the DSP has detected both interrupts, transmit and receive, it will shift the new data in the receive buffer into the inactive half of the transmit buffer.

It will continue this loop-back process until the user presses a key and an interrupt from the UART signals the process to stop.

### 5.3 Testing the external SRAM

If this memory test is selected, the DSP will start to write and read data to the memory space defined in the configuration file as 'testmem' and 'testprg'. Each stored value will be checked for errors and any faults will increment the error counter 'ERR'. The program will stay in this mode until the user presses any key - at which point, the memory test will stop and the actual value of ERR will be displayed.

At the start of the routine the contents of the memory locations in the data space specified by 'testmem' will be set to zero. The contents of the data space will then be copied to the program memory specified by 'testprg'.

After that, the DSP reads a location in the program space, checks if the value is correct and stores a new value (0x5555) in this cell. If the value read is different from the value expected, the counter ERR is incremented. The same task is performed in data space. This procedure is repeated for the whole length of the memory defined by 'RANGE' by incrementing the address pointers by 1 after each store instruction.

The same test is repeated with 0xaaaa, 0xffff and 0x0000 as the new values are written to the external memory locations.

The consecutive values are chosen to maximize the number of data bits changed with each access.

The test is executed 16 times for each of the four values. The LED indicates the current number of iterations performed (0 to 16).

The program then starts a different test.

The DSP executes 8 consecutive writes to the external data space. The data written alternates between 0x5555 and 0xaaaa with each cycle. After this, the DSP reads out this data and stores the sum of the eight values in the accumulator to check it. If the result is not correct, ERR is incremented. This procedure is repeated for the following blocks of eight words until the DSP reaches the end of the external memory space, defined by RANGE.

After this, the program executes the same routine with 0x0000 and 0xffff as the values to be written.

Finally the DSP restarts the whole memory test without resetting ERR until the program detects an interrupt from the UART, signaling that a key has been pressed by the user.

## References

1. Designing with the TLC320AC01 Analog Interface for DSPs, John Walliker and Julian Daley, Texas Instruments 1994, SLAAE09
2. TLC320AC01 Analog Interface circuit, Texas Instruments 1993, SLAS057A
3. TMS320C5x, TMS320LC5x data sheet, Texas Instruments 1995, SPRS030A
4. TMS320C5x User's Guide, Texas Instruments 1993, SPRU56C
5. Power Supply Circuits, Texas Instruments 1996, SLVD002
6. Low-Voltage Logic, Texas Instruments 1994, SCVDE03
7. TMS320C54x User's Guide, Texas Instruments 1995, SPRU131A
8. TMS320C54x Data Sheet, Texas Instruments 1996, SPRS039
9. TMS320C54x Serial Ports User's Guide, Texas Instruments 1995, SPRU156
10. Digital Design Workshop, E. Haseloff and P. Forstner, Texas Instruments 1995

## Appendix A Source Code

```

/*****
/*          FILE NAME: MAIN.C          */
/*          PROGRAM UART COMMUNICATION WITH TMS320LC56          */
/*          */
/*****

#define clean_up receive_data(); break /* cleanup macro          */
#define maxstring 40
#include "main1.c"          /* flag names, constants          */
extern void memory();
extern void ac01();
extern int ERR;          /* error counter of memtest          */
int DIMR = 6;
/*_____*/

void main(void)
{
#define max 100
char C, COMMAND;
char s[max];
int i,n;
int DONE = 0;
asm(" splk #1000100010111000b, PMST ");
asm(" splk #01C0h, PDWSR");          /* reset waitstates, 7 FOR IO          */
asm(" LMMR IMR,#_DIMR");
asm(" splk #0ffffh,IFR");          /* clear all pending int          */
asm(" CLRC INTM");          /* disable interrupts          */
port8003 = 135;          /*load value for line cntrl reg,enable DLAB          */
/*          10000111          */
/*          |||||++----- word length = 8bit          */
/*          |||||+----- number of stop bits = 2          */
/*          |||||+----- parity bit = no          */
/*          |||||+----- even parity select = odd          */
/*          |||||+----- stick parity bit = disabled          */
/*          |||||+----- break control bit = disabled          */
/*          |||||+----- divisor latch bit = enabled          */
port8000 = 16;          /* load divisor with 16 -> 9600 Baud          */
port8101 = 0; /* clear upper byte of divisor          */
port8003 = 7; /* disable DLAB bit          */
C = '\n';          /* load C with a character          */
port8000 = C; /* send this character to UART to activate a          */
/*          /* transmit interrupt          */
C = port8000; /* read received word from the UART to claeir it          */

sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendstring(" *** LC56 test platform ***",27);
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendstring("Please make your choice... ",27);
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendstring("Memorytest (m)",21);
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendstring("AC01 in Loop Back (a)",21);
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */
sendstring("Stop (s)",21);
sendinteger(0xa);          /* reset cursor          */
sendinteger(0xd);          /* line feed          */

```



```

while(!DONE)          /* stay in following loop while DONE = 0          */
{
  COMMAND = receivechar();      /* load COMMAND with character received f. UART*/
  sendchar(COMMAND);           /* write character received on the monitor */
  sendinteger(0xa);            /* reset cursor */
  sendinteger(0xd);            /* line feed */
  switch (COMMAND)             /* execute the command... */
  {
    case 'M': case 'm': memory();          /* run function memory */
      printresult(ERR);                  /* print error counter */
      sendinteger(0xa);                  /* reset cursor */
      sendinteger(0xd);                  /* line feed */
      sendstring("New choice? (m,a or s)",22);
      break;                             /* return from 'switch' */

    case 'a': case 'A': sendstring("ac01 in loop back mode",22);
      sendinteger(0xa);                  /* reset cursor */
      sendinteger(0xd);                  /* line feed */
      sendstring("press any key to stop",21);
      sendinteger(0xa);                  /* reset cursor */
      sendinteger(0xd);                  /* line feed */
      ac01();                            /* run function ac01 */
      sendstring("New choice? (m,a or s)",22);
      break;                             /* return from 'switch' */

    case 'S': case 's': sendstring("program completed",17);
      sendinteger(0xa);                  /* reset cursor */
      sendinteger(0xd);                  /* line feed */
      DONE = 1;                          /* set DONE to 1 */
      break;                             /* return from 'switch' */

  }
}
}
/*

```

```

/*****
/*                               FILE NAME: main1.C                               */
/*                               LC56 UART COMMUNICATION FUNCTIONS                     */
/*                               */
/* clear_messages() receive_data() send_data() send_file_data() */
/* initialize_evm() receive_command() send_command() receive_file_data() */
/* printresult() */
/*****
#include <string.h>
/*#include "evml.h" */ /* flag names, constants */
#include <ioports.h>
extern int RXREADY; /* indicator for UART receive intrpt */
extern int TXREADY; /* indicator for UART transmit intrpt */
unsigned char reply;
ioport char port8000; /* UART transmit & receive buffer */
ioport int port8003; /* UART Line Control Register */
ioport int port8100; /* UART Divisor lower byte */
ioport int port8101; /* UART Divisor upper byte */
/*
-----

void sendstring(char s[], int lim)
{
    int c,i;
    i = 0;
    while(--lim >= 0) /* send lim characeters of string s */
        sendchar(s[i++]);
}
/*
-----

void sendchar(char c)
{
    while(!TXREADY); /* wait for txready interrupt */
    port8000 = c; /* write char to UART's TX register */
    TXREADY = 0; /* Clear ready to transmit bit */
}
/*
-----

char receivechar()
{
    RXREADY = 0;
    reply = port8000; /* read char from UART's RX register */
    while(!RXREADY); /* wait for txready interrupt */
    reply = port8000; /* read char from UART's RX register */
    RXREADY = 0; /* Clear ready to receive bit */
    return(reply); /* return the received value */
}
/*
-----

void sendinteger(int i)
{
    while(!TXREADY); /* wait for txready interrupt */
    port8100 = i; /* write char to UART's TX register */
    TXREADY = 0; /* Clear ready to transmit bit */
}
/*
-----

```

```

void itoa(int n,char s[])          /* transform the integer value n into  */
{                                  /* the ASCII string s          */
    int i, sign;
    if ((sign = n) < 0)
        n = -n;
    i = 0;
    do {
        s[i++] = n % 10 + '0';
    }
    while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
/*_____*/

void reverse(char s[])           /* reverses the string e.g. 'abc' -> 'cba' */
{
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
        {
            c = s[i];
            s[i] = s[j];
            s[j] = c;
        }
}
/*_____*/

void printresult(int E)
{
    int n, i;
    long L;
    char s[10];
    sendstring("Memorytest identified ",22);
    L = E;          /* load E in an long value          */
    n = ltoa(L,s); /* transform L in ASCII string, n=length of s */
    for(i=0;i<n;i++) /* send this string          */
        {
            sendchar(s[i]);
        }
    sendstring(" errors",7);
    sendinteger(0xa); /* line feed          */
}

```





```
rsvd1C b    rsvd1C
rsvd1E b    rsvd1E
rsvd20 b    rsvd20
trap  b     trap
nmi   b     nmi
rsvd26 b    rsvd26
rsvd28 b    rsvd28

        .text
        .include "ac01.asm"
        .include "memory.asm"

; interrupt handlers

getxdata:    splk    #1, gotxflag    ; set flag
             rete    ; return form interrupt restoring
getrdata:    splk    #1, gotrflag    ; set flag
             rete    ; return form interrupt restoring

rxr:        LACC    #1                ; set RXREADY if
            SAMP    _RXREADY        ; int2 occurs
            RETE

txr:        LACC    #1                ; set TXREADY if
            SAMP    _TXREADY        ; int3 occurs
            RETE

.end
```

```

*****
*                               TMS320CL56                               *
*                               function memory test                       *
*                               include file of mem.asm                     *
*****
.text                               ; start of main program
_ac01:
  POPD    *+                          ; save stack pointer
  SAR     AR0,*+                        ; to be compatible with
  SAR     AR1,*                          ; routines written in C
  LARK    AR0,1
  LAR     AR0,*0+

  ldp     #0                            ; set dp to first page
  setc    INTM                          ; disable interrupts
  setc    SXM                            ; set sign ext mode
  setc    OVM                            ; set Overflow mode

; The code below does the following: disables visibility of address
; bus, on chip single access ram is mapped into prog space,sets the IPTR
; to point to 8000h and selects MICROPROCESSOR mode
  clrc    xf
  splk    #1000100010111000b, PMST
  clrc    CNF                            ; B0 mapped in data space

; Set the interrupt mask to respond to buffered serial interrupt
  splk    #0c6h,IMR                      ; only respond to buffered serial rx & tx int and
                                          ; uart
  splk    #08h,BSPC                       ; reset and configure BSPC
  splk    #0b400h,BSPCE                   ; configure BSPCE, buf for tx/rx enabled
                                          ; 16bits ext frame & clk
  splk    #XTOP, AXR                      ; load startaddress of buffer in AXR
  splk    #RTOP, ARR                      ; load startaddress of buffer in ARR
  splk    #16, BKX                        ; load buf size in BKX (only for initialisation)
  splk    #XSIZE, BKR                     ; load buf size in BKR
  splk    #0c0h, IFR                      ; clear any pending interupt

  setc    xf                              ; release codec
  rpt     #20
  nop

; Reset the codec and release it again to make it ignore first garbage
; word generated by serial port
  clrc    xf                              ; reset codec
  rpt     #20                              ; wait for 20 cycles
  nop
  setc    xf                              ; release codec
  splk    #048h, BSPC                     ; start SPI transmit part
  CLRC    INTM                            ; enable interrupts

; Serial interface and AC01 are now in stable state
; set up AC01
  waitxint                                ; wait for xint, ac01 has received initialization
  splk    XTOP, AXR                       ; reset tx buffer address
  splk    #088h, BSPC                     ; start SPI receive part
  splk    #XSIZE, BKX                     ; enable full size of tx buffer
  waitrint                                ; wait for rint after half buffer is full
  splk    RTOP, ARR                       ; reset rx buffer address

  LAR     AR4, #0                          ; Loop counter for debugging
LOOP    LAR     AR3, #HSIZE                 ; load loop counter with 'half buffersize' - 1
  MAR     *,AR3
  splk    #RTOP, TEMP1
  LACL    TEMP1                            ; load address of lower half of receive buffer
  BIT     BSPCE,1                          ; check RH flag in BSPCE register
  BCND    LRBP, NTC                        ; branch if RH = 0
  ADD     #HSIZE                            ; add half buffer size if RH = 1
LRBP    SACL    TEMP1
  LAR     AR5, TEMP1                       ; load receive buffer pointer
  splk    #XTOP, TEMP1
  LACL    TEMP1                            ; load address of lower half of transmit buffer
  BIT     BSPCE,4                          ; check XH flag in BSPCE register
  BCND    LXBP, NTC                        ; branch if XH = 0
  ADD     #HSIZE                            ; add half buffer size if RH = 1
LXBP    SACL    TEMP1

```

```
LAR    AR2, TEMP1      ; load transmit buffer pointer
MAR    *-,AR5          ; subtract 1 from loop counter AR3

SHIFT  LACL    *+, AR2      ; activate tx buffer pointer
        SACL    *+, AR3      ; store 1st rsample as 1st txsample
        BANZ   SHIFT,*-,AR5  ; repeat 'half buffer size' times
        NOP
        NOP
        splk   #0b400h, BSPCE ; reload SPCE and set BXE again
        splk   #0c8h, BSPC   ; start SPI rx and tx part

        waitxint           ; wait for last samples transmitted
        waitrint          ; wait for new samples received

        LACC   ARR
        LACC   AXR
        LACC   BKR
        LACC   BKX
        MAR    *, AR4      ; increment loop counter for
        MAR    *+, AR5     ; debugging
        LACC   _RXREADY    ; check if interrupt from the UART
        BCND   LOOP,EQ     ; occured. If not stay in the loop

        NOP
        nop

        LARP   AR1        ; reload stack pointer
        SBRK   2
        LAR    AR0,*-
        PSHD   *
        RET
```



```

*****
*                               TMS320CL56                               *
*                               function memory test                       *
*                               include file of mem.asm                     *
*****
        .text
_memory:
        POPD    *+                ; save stack pointer
        SAR     AR0,*+            ; to be compatible with
        SAR     AR1,*            ; routines written in C
        LARK    AR0,1
        LAR     AR0,*0+

        ldp     #0
        splk    #0, greg
        splk    #1000100010111000b, PMST
        splk    #111000000b, PDWSR        ; set software wait states
;          |||||+----- program: 0 wait states
;          |||||+----- data: 0 wait states
;          +++----- i/o: 7 wait states

        CLRC    INTM                ; enable interrupts
        MAR     *,AR3                ; activate AR3
        CLRC    CNF                  ; map on-chip RAM into data space
        LACL    #0                    ; reset the
        SACL    _ERR                  ; Error counter
        LAR     AR3,#CLOCK            ; preload counter values
        SACL    *+                    ; in data space locations
        LACL    #1
        SACL    *+
        LACL    #2
        SACL    *+
        LACL    #3
        SACL    *+
        LACL    #4
        SACL    *+
        LACL    #5
        SACL    *+
        LACL    #6
        SACL    *+
        LACL    #7
        SACL    *+
        LACL    #8
        SACL    *+
        LACL    #9
        SACL    *+
        LACL    #10
        SACL    *+
        LACL    #11
        SACL    *+
        LACL    #12
        SACL    *+
        LACL    #13
        SACL    *+
        LACL    #14
        SACL    *+
        LACL    #15
        SACL    *+

        ldp     #0                ; set dp to first page
        CLRC    SXM                ; suppress sign extension

START   LAR     AR6,#testmem        ; load start address of
        MAR     *, AR6                ; data memory to test
        RPTZ    #RANGE                ; load 0 in all memory
        SACL    *+                    ; locations within RANGE
        LAR     AR6,#testmem        ; reset also program space
        LACC    #tstprg                ; while copy data to program
        RPT     #RANGE
        TBLW    *+

```

```

MAR    *,AR3                ; ARP points to AR3
LAR    AR3,#CLOCK           ; Load AR3 with start value
LAR    AR2,#15              ; AR2 is the loop counter
LAR    AR5,#0

LOOPM:
OUT     *,,0001h            ; Output value of AR3

LACC   #0h
LAR    AR6,#testmem        ; load start address of
MAR    *,AR6                ; data memory to test
LAR    AR7,#RANGE          ; load counter with the length
                                ; of memory to test
LACC   #tstprg              ; store the start address or
SACL   ADDR                 ; program memory to test in ADDR

OTOA:  LACC   ADDR           ; read first program memory
TBLR   TEMP                 ; location
LACC   #0h
XOR    TEMP                 ; check if value is correct
BCND   ERR_P1,NEQ           ; if not correct then ERR_P1
NOP
NOP

P1:    LACC   #0AAAAh        ; load new value to store in
SACL   TEMP                 ; program memory location
LACC   ADDR                 ; store this value in prog
TBLW   TEMP
ADD    #1                    ; increment ADDR
SACL   ADDR
LACC   #0h
XOR    *                     ; read data memory location
                                ; and check value
BCND   ERR_D1,NEQ           ; if not correct ERR_P1
NOP
NOP

D1:    LACC   #0AAAAh        ; store new value in
SACL   *,,AR7               ; data memory location
BANZ   OTOA,*-,AR6          ; stay in loop if loop counter
                                ; AR7 is not 0. Post decrement AR7
NOP
NOP

LACC   #tstprg              ; same as OTOA but change values
LAR    AR7,#RANGE           ; from A to 5
LAR    AR6,#testmem
SACL   ADDR
ATO5:  LACC   ADDR
TBLR   TEMP
LACC   #0AAAAh
XOR    TEMP
BCND   ERR_P2,NEQ
NOP
NOP

P2:    LACC   #5555h
SACL   TEMP
LACC   ADDR
TBLW   TEMP
ADD    #1
SACL   ADDR
LACC   #0Ah
XOR    *
BCND   ERR_D2,NEQ
NOP
NOP

D2:    LACC   #5555h
SACL   *,,AR7
BANZ   ATO5,*-,AR6
NOP
NOP

```

```

        LAR    AR6,#testmem      ; same as OTOA but change values
        LACC   #tstprg          ; from 5 to F
        LAR    AR7,#RANGE
        SACL   ADDR
f5TOF:  LACC   ADDR
        TBLR   TEMP
        LACC   #5555h
        XOR    TEMP
        BCND   ERR_P3,NEQ
        NOP
        NOP
P3:     LACC   #0FFFFh
        SACL   TEMP
        LACC   ADDR
        TBLW   TEMP
        ADD    #1
        SACL   ADDR
        LACC   #5555h
        XOR    *
        BCND   ERR_D3,NEQ
        NOP
        NOP
D3:     LACC   #0FFFFh
        SACL   *+,AR7
        BANZ   f5TOF,*-,AR6
        NOP
        NOP

        LAR    AR6,#testmem      ; same as OTOA but change values
        LACC   #tstprg          ; from F to 0
        LAR    AR7,#RANGE
        SACL   ADDR
F7TOO:  LACC   ADDR
        TBLR   TEMP
        LACC   #0FFFFh
        XOR    TEMP
        BCND   ERR_P4,NEQ
        NOP
        NOP
P4:     LACC   #0h
        SACL   TEMP
        LACC   ADDR
        TBLW   TEMP
        ADD    #1
        SACL   ADDR
        LACC   #0FFFFh
        XOR    *
        BCND   ERR_D4,NEQ
        NOP
        NOP
D4:     LACC   #0h
        SACL   *+,AR7
        BANZ   F7TOO,*-,AR6
        NOP
        NOP

        MAR    *,AR2
        BANZ   LOOPM,*-,AR3      ; Branch to start of loop
        NOP
;Start this part after 16 times LOOPM

; The following test will generate consecutive writes and reads
; to the external memory
        LACC   #RANGE, 13        ; load range divided by 8 in AR3
        SACL   TEMP
        LAR    AR3, TEMP
        LAR    AR4, #testmem     ; load start address of datamem
        MAR    *,AR4             ; to test in AR4
CONS1:  LACC   #05555h          ; load accumulator with
        SACL   TEMP              ; 5555aaaaah
        LACC   TEMP, 16
        OR    #0aaaaah

```

```

SACL  *+                ; store aaaah in external data
SACH  *+                ; store 5555h in external data
SACL  *+                ; store aaaah in external data
SACH  *+                ; store 5555h in external data
SACL  *+                ; store aaaah in external data
SACH  *+                ; store 5555h in external data
SACL  *+                ; store aaaah in external data
SACH  *+                ; store 5555h in external data
SAR   AR4,ADDR         ; reset the address pointer
LACL  ADDR
SUB   #8
SACL  ADDR
LAR   AR4, ADDR
LACC  *+                ; read and accumulate the values
ADD   *+                ; written before
ADD   *+
ADD   *+
ADD   *+
ADD   *+
ADD   *+
ADD   *+
ADD   *+
ADD   *+
ADD   #4                ; add 4 to the accumulator
AND   #0ffffh          ; if all reads and writes are ok
BCND  ERR_CONS1, NEQ   ; ACCU = 0. If not ERR_CONS1
C1:   MAR   *, AR3      ; stay in loop until AR3 = 0
      BANZ  CONS1,*-,AR4 ; post increment loop counter AR3

      LACC  #RANGE, 13   ; same as above
      SACH  TEMP        ; this time with 0000 and ffffh
      LAR   AR3, TEMP    ; as values to be written to
      LAR   AR4, #testmem ; external memory
CONS2: LACC  #0ffffh
      SACL  TEMP
      LACC  TEMP, 16
      SACL  *+
      SACH  *+
      SACL  *+
      SACH  *+
      SACL  *+
      SACH  *+
      SACL  *+
      SACH  *+
      SAR   AR4,ADDR
      LACL  ADDR
      SUB   #8
      SACL  ADDR
      LAR   AR4, ADDR
      LACC  *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   *+
      ADD   #4
      AND   #0ffffh
      BCND  ERR_CONS2, NEQ
C2:   MAR   *, AR3
      BANZ  CONS2,*-,AR4

      LACC  _RXREADY    ; check if interrupt occurred
      BCND  START,EQ    ; from UART. If not start again
      NOP   ; with memory test
      NOP
      LARP  AR1          ; reload stack pointer
      SBRK  2
      LAR   AR0,*-
      PSHD  *
      RET

```

```
ERR_D1 LACC _ERR          ; add 1 to the
      ADD #1              ; error counter
      SACL _ERR          ; and continue
      B D1               ; program execution
ERR_D2 LACC _ERR
      ADD #1
      SACL _ERR
      B D2
ERR_D3 LACC _ERR
      ADD #1
      SACL _ERR
      B D3
ERR_D4 LACC _ERR
      ADD #1
      SACL _ERR
      B D4
ERR_P1 LACC _ERR
      ADD #1
      SACL _ERR
      B P1
ERR_P2 LACC _ERR
      ADD #1
      SACL _ERR
      B P2
ERR_P3 LACC _ERR
      ADD #1
      SACL _ERR
      B P3
ERR_P4 LACC _ERR
      ADD #1
      SACL _ERR
      B P4
ERR_CONS1 LACC _ERR
      ADD #1
      SACL _ERR
      B C1
ERR_CONS2 LACC _ERR
      ADD #1
      SACL _ERR
      B C2
```

```

/*****
/* Test Program for EMU5X running on a LC56
/* Configurations of processor:
/* 1) Microprocessor mode (MP/MC=1)
/* 2) B0 DARAM configure on data space (CNF=0)
/* 3) 6K Single Access Ram configure in Program space
*****/

main.obj
mem.obj
-stack 512
-cr
-i c:\dsp\c5x\lib
-l rts50.lib

-m main.map
-o main.out
-v0

MEMORY
{
    PAGE 0 :
        EXT_PROG : origin = 555h ,length = 7000h /* 32K ext RAM */
        PROG_RAM : origin = 8800h ,length = 1000h /* 6K RAM */

    PAGE 1 :

        REGS : origin = 0000h ,length = 0060h /* MEM REG */
        DARAM_B2: origin = 0060h ,length = 0020h /* Always Data */
        RAM_B0B1: origin = 0100h ,length = 0400h /* CNF=0 */
        EXT_DATA: origin = 2000h ,length = 6000h /* 24K ext RAM */
        BLK0 : origin = 0800h ,length = 0800h /* OVLY=1 */
}

/*-----*/
/* SECTIONS ALLOCATION */
/*-----*/

SECTIONS
{
    test_p : { } > EXT_PROG PAGE 0
    vectors: { } > PROG_RAM PAGE 0
    .text : { } > PROG_RAM PAGE 0 /* CODE */
    .bss : { } > RAM_B0B1 PAGE 1 /* GLOBAL VARS,STACK,HEAP */
    .data : { } > RAM_B0B1 PAGE 1 /* Initialized var */
    test_d : { } > EXT_DATA PAGE 1
    b2 : { } > DARAM_B2 PAGE 1 /* DARAM_B2 */
    .cinit : { } > PROG_RAM PAGE 0
    .const : { } > RAM_B0B1 PAGE 1
    .stack : { } > RAM_B0B1 PAGE 1
    txbuff : { } > BLK0 PAGE 1 /* Transmit Buffer */
    rxbuff : { } > BLK0 PAGE 1 /* Receive Buffer */
}

```

## Appendix B PAL Programming

" EVM used for TMS320LC56

```

MODULE module_name
    lc56_2 DEVICE 'p16v8r';

"INPUTS
    CLK,A15,A14,RW,IS          PIN 1,2,3,4,5;
    DS,CLKOUT,STRB,RD         PIN 6,7,8,9;

    OE                        PIN 11;

"OUTPUTS
    CLK_A                     PIN 19;
    WEX                       PIN 18;
    BE                        PIN 17;
    ISX                       PIN 16;
    A                         PIN 14;
    RDX                       PIN 13;
    EX                        PIN 12;
    X,C,Z=.X.,.C.,.Z.;

EQUATIONS

"UART READ & WRITE ACCESS

!CLK_A = CLKOUT;

!A := !RD;

!RDX = !IS & !STRB & RW & !A & !A14 & A15;
!WEX = !IS & !STRB & !RW & !A14 & A15;

"LED STROBE

!ISX = !A15 & !A14 & !IS & !STRB;

"BUFFER ENABLE

!BE = !IS # !DS & A15;

"EPROM ENABLE

!EX = !IS & A15 & A14 # !DS & A15;

```

```
TEST_VECTORS ([CLK,RD,CLKOUT] -> [CLK_A,A])
    [C, 0, 0] -> [ 1, 0];
    [C, 1, 1] -> [ 0, 1];

TEST_VECTORS ([CLK,RD,IS,STRB,RW,A14,A15] -> [A,RDX])
    [C, 0, 0, 0, 0, 0, 1] -> [0, 1 ];
    [C, 0, 0, 0, 1, 0, 1] -> [0, 0 ];
    [C, 1, 0, 0, 1, 0, 1] -> [1, 1 ];
    [C, 0, 0, 1, 1, 0, 1] -> [0, 1 ];
    [C, 0, 0, 0, 1, 0, 0] -> [0, 1 ];

TEST_VECTORS ([CLK,IS,STRB,RW,A14,A15] -> [WEX])
    [C, 0, 0, 0, 0, 1] -> [ 0 ];
    [C, 1, 0, 0, 0, 1] -> [ 1 ];
    [C, 0, 1, 0, 0, 1] -> [ 1 ];
    [C, 0, 0, 1, 0, 1] -> [ 1 ];
    [C, 0, 0, 0, 1, 1] -> [ 1 ];

TEST_VECTORS ([CLK,IS,A15,A14,STRB] -> [ISX])
    [C, 1, 0, 0, 0 ] -> [ 1 ];
    [C, 0, 0, 0, 0 ] -> [ 0 ];
    [C, 0, 0, 1, 0 ] -> [ 1 ];
    [C, 0, 1, 0, 0 ] -> [ 1 ];
    [C, 0, 1, 1, 0 ] -> [ 1 ];
    [C, 0, 0, 0, 1 ] -> [ 1 ];
TEST_VECTORS ([CLK,IS,A15,DS] -> [BE])
    [C, 1, 1, 0] -> [ 0];
    [C, 0, 0, 0] -> [ 0];
    [C, 1, 0, 0] -> [ 1];
    [C, 1, 0, 1] -> [ 1];

TEST_VECTORS ([CLK,A15,A14,IS,DS] -> [EX])
    [C, 1, 1, 0, 1] -> [ 0 ];
    [C, 1, 0, 0, 0] -> [ 0 ];
    [C, 0, 1, 0, 0] -> [ 1 ];
    [C, 0, 0, 1, 0] -> [ 1 ];
    [C, 0, 0, 0, 1] -> [ 1 ];

END module_name
```



```

" EVM used for TMS320LC541,3,6

MODULE module_name
    lead_1 DEVICE 'p16v8r';

" INPUTS
    CLK,A15,A14,RW,IS          PIN 1,2,3,4,5;
    DS,CLKOUT,STRB             PIN 6,7,8;
    MSTRB                       PIN 15;
    OE                          PIN 11;

" OUTPUTS
    CLK_A                       PIN 19;
    WEX                         PIN 18;
    BE                          PIN 17;
    ISX                         PIN 16;
    WE                          PIN 14;
    RDX                         PIN 13;
    EX                          PIN 12;
    X,C,Z=.X.,.C.,.Z.;

EQUATIONS

"UART READ & WRITE ACCESS

!CLK_A = CLKOUT;

!WE     = !MSTRB & !RW;

!RDX    = !IS & !STRB & RW & !A14 & A15;

!WEX    = !IS & !STRB & !RW & !A14 & A15;

"LED STROBE

!ISX    = !A15 & !A14 & !IS & !STRB;

"BUFFER ENABLE

!BE     = !IS # !DS & A15;

"EPROM ENABLE

!EX     = !IS & A15 & A14 # !DS & A15;

```

```
TEST_VECTORS ([CLK,CLKOUT] -> [CLK_A])
    [C,    0] -> [ 1];
    [C,    1] -> [ 0];

TEST_VECTORS ([CLK,MSTRB, RW] -> [WE])
    [C,    0 ,  1] -> [ 1];
    [C,    1 ,  1] -> [ 1];
    [C,    1 ,  1] -> [ 1];
    [C,    0 ,  0] -> [ 0];

TEST_VECTORS ([CLK,IS,STRB,RW,A14,A15] -> [RDX])
    [C,  0,  0,  0,  0,  1] -> [ 1 ];
    [C,  0,  0,  1,  0,  1] -> [ 0 ];
    [C,  0,  0,  1,  0,  1] -> [ 0 ];
    [C,  0,  1,  1,  0,  1] -> [ 1 ];
    [C,  0,  0,  1,  0,  0] -> [ 1 ];

TEST_VECTORS ([CLK,IS,STRB,RW,A14,A15] -> [WEX])
    [C,  0,  0,  0,  0,  1] -> [ 0 ];
    [C,  1,  0,  0,  0,  1] -> [ 1 ];
    [C,  0,  1,  0,  0,  1] -> [ 1 ];
    [C,  0,  0,  1,  0,  1] -> [ 1 ];
    [C,  0,  0,  0,  1,  1] -> [ 1 ];

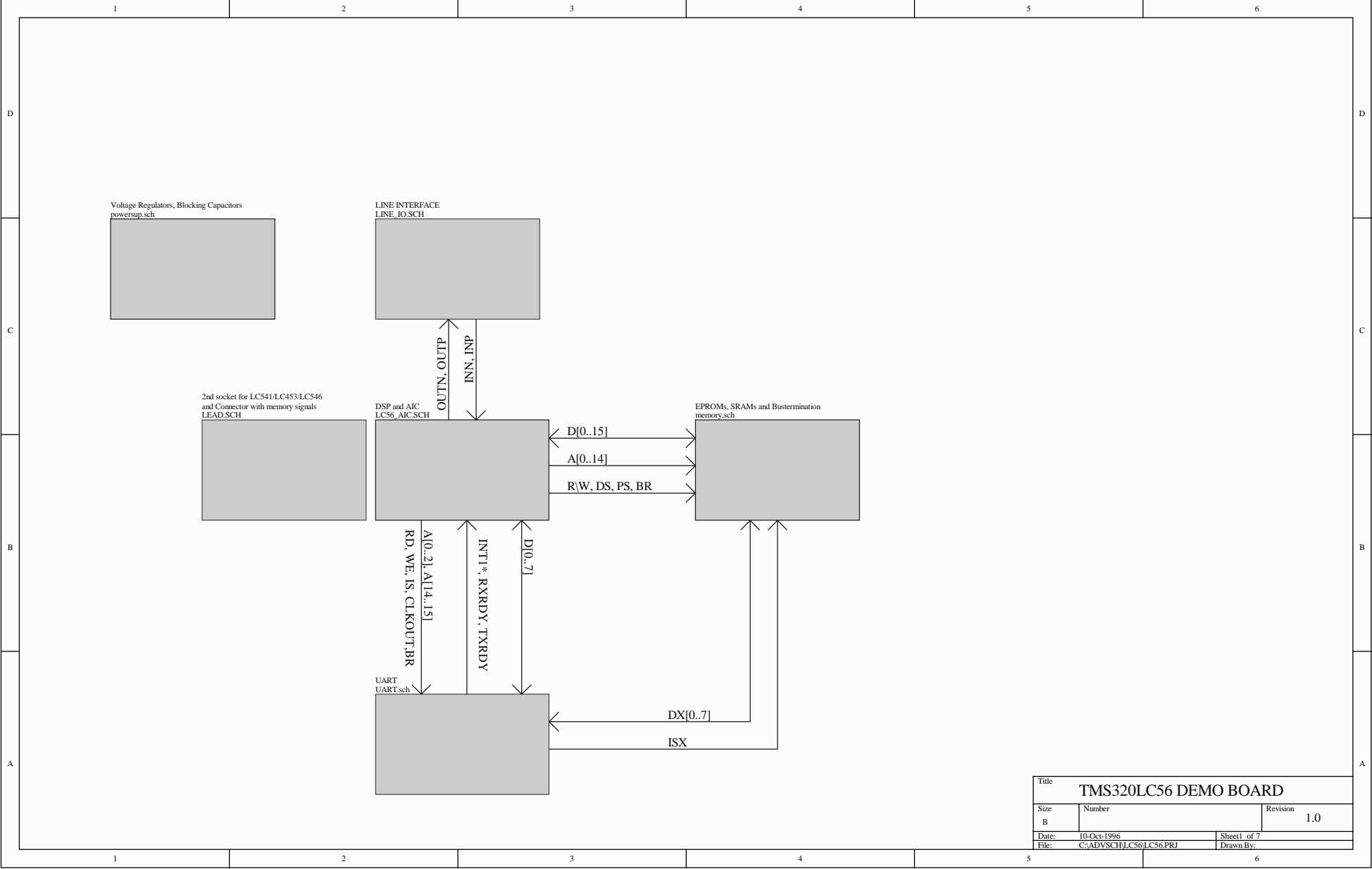
TEST_VECTORS ([CLK,IS,A15,A14,STRB] -> [ISX])
    [C,  1,  0,  0,  0 ] -> [ 1 ];
    [C,  0,  0,  0,  0 ] -> [ 0 ];
    [C,  0,  0,  1,  0 ] -> [ 1 ];
    [C,  0,  1,  0,  0 ] -> [ 1 ];
    [C,  0,  1,  1,  0 ] -> [ 1 ];
    [C,  0,  0,  0,  1 ] -> [ 1 ];

TEST_VECTORS ([CLK,IS,A15,DS] -> [BE])
    [C,  1,  1,  0] -> [ 0];
    [C,  0,  0,  0] -> [ 0];
    [C,  1,  0,  0] -> [ 1];
    [C,  1,  0,  1] -> [ 1];

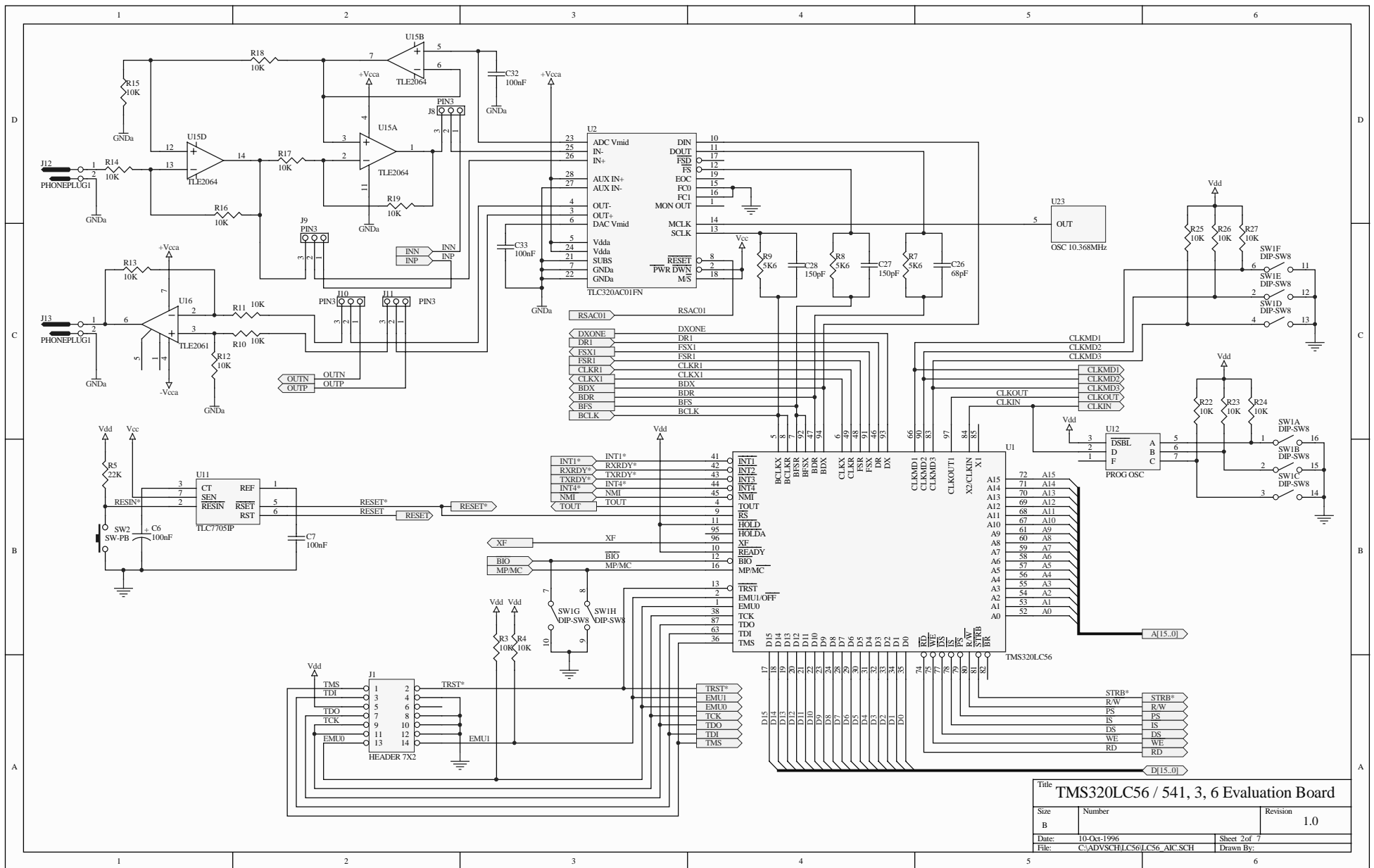
TEST_VECTORS ([CLK,A15,A14,IS,DS] -> [EX])
    [C,  1,  1,  0,  1] -> [ 0 ];
    [C,  1,  0,  0,  0] -> [ 0 ];
    [C,  0,  1,  0,  0] -> [ 1 ];
    [C,  0,  0,  1,  0] -> [ 1 ];
    [C,  0,  0,  0,  1] -> [ 1 ];

END module_name
```

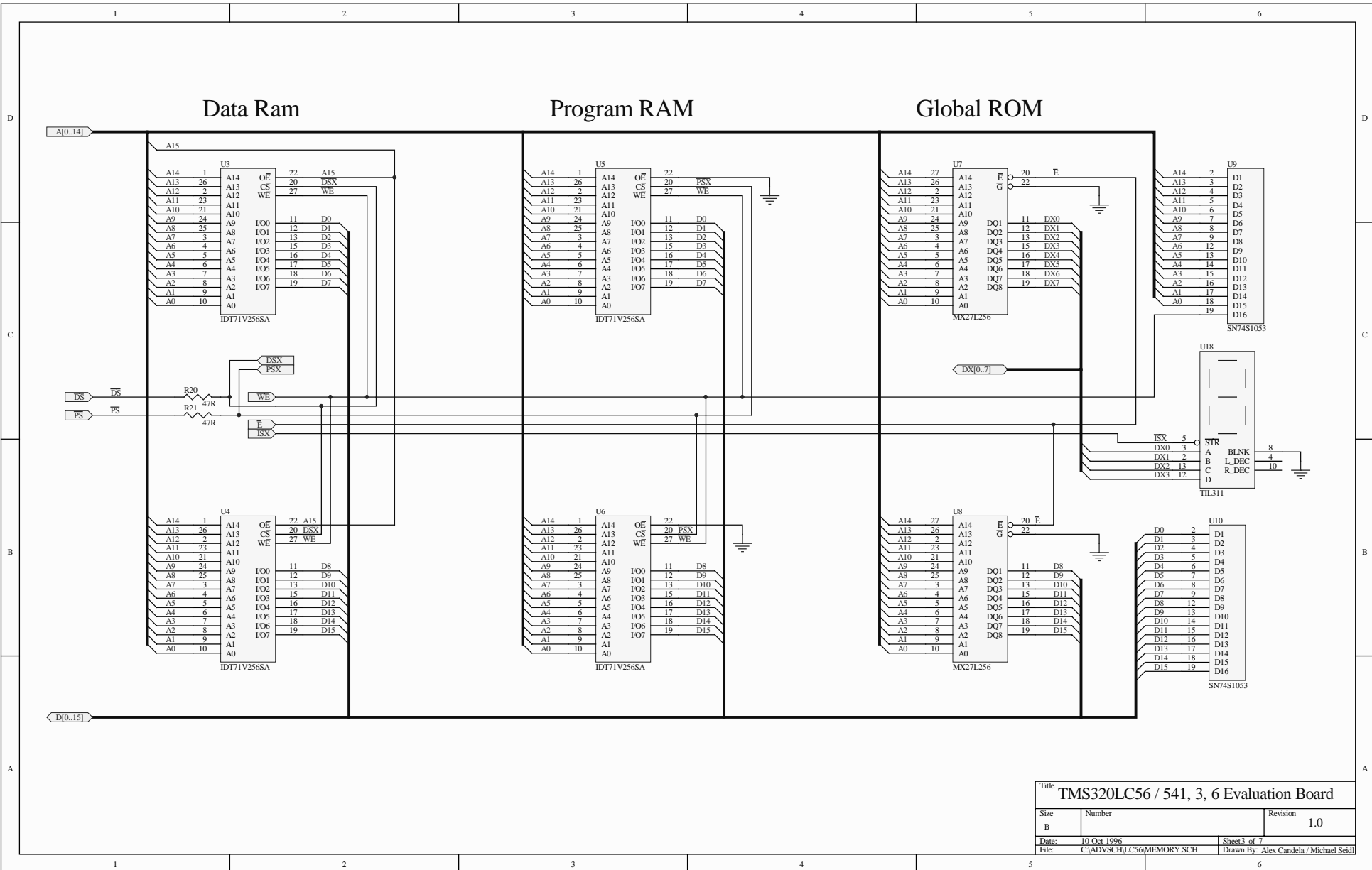
## Appendix C Schematics



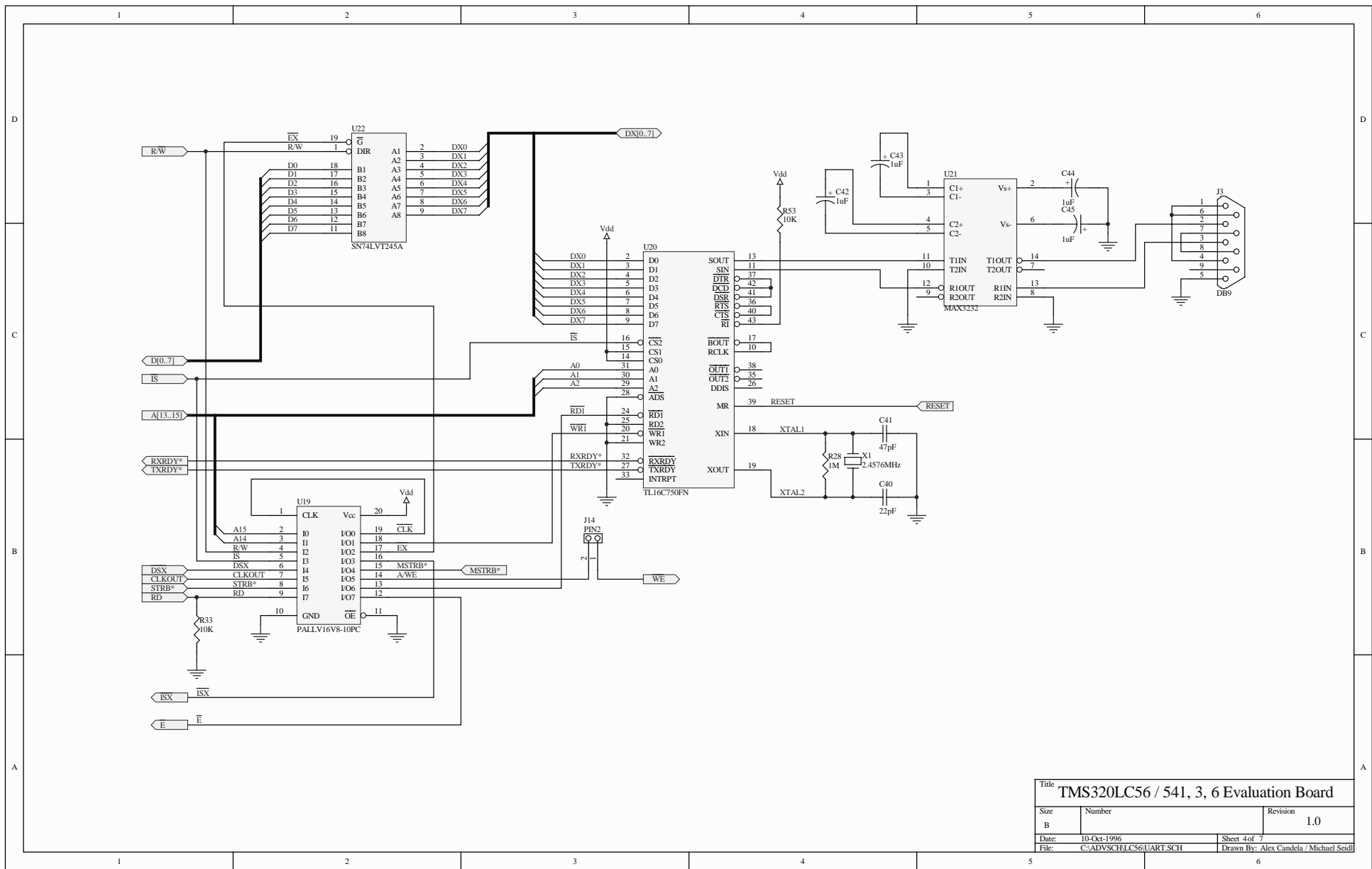
Title			TMS320LC56 DEMO BOARD		
Size	Number	Revision		1.0	
B					
Date:	10-Oct-1996	Sheet		1 of 7	
File:	C:\ADV\SCH\LC56\LC56.PRJ	Drawn By:			



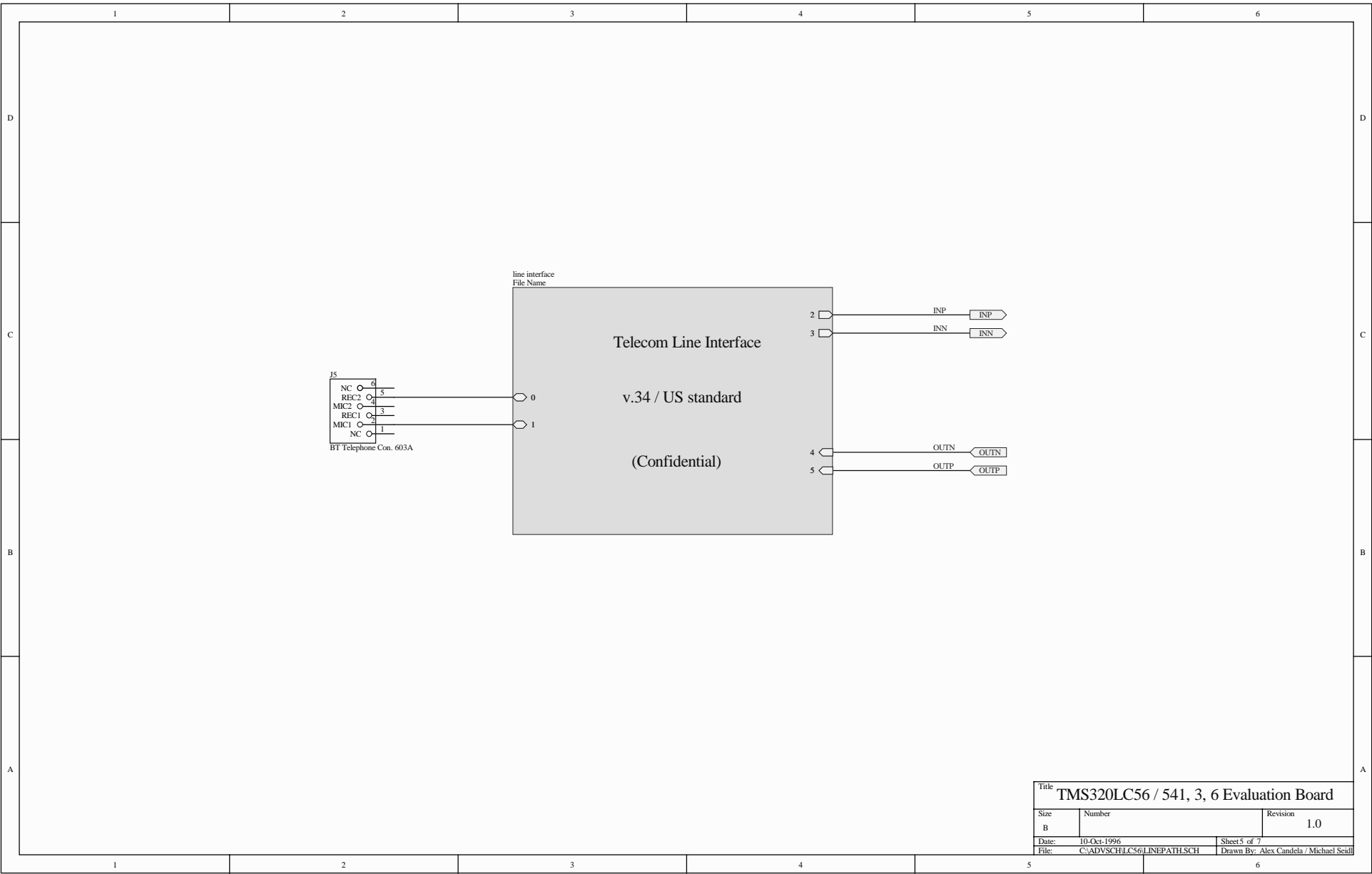
Title			TMS320LC56 / 541, 3, 6 Evaluation Board		
Size	Number	Revision		1.0	
B					
Date:	10-Oct-1996	Sheet 2 of 7		Drawn By:	
File:	C:\ADVSCH\TMS320LC56_AIC.SCH				



Title			TMS320C56 / 541, 3, 6 Evaluation Board		
Size	Number			Revision	1.0
Date:		10-Oct-1996		Sheet 3 of 7	
File:		C:\ADV\SCH\LC56\MEMORY.SCH		Drawn By: Alex Candela / Michael Scall	

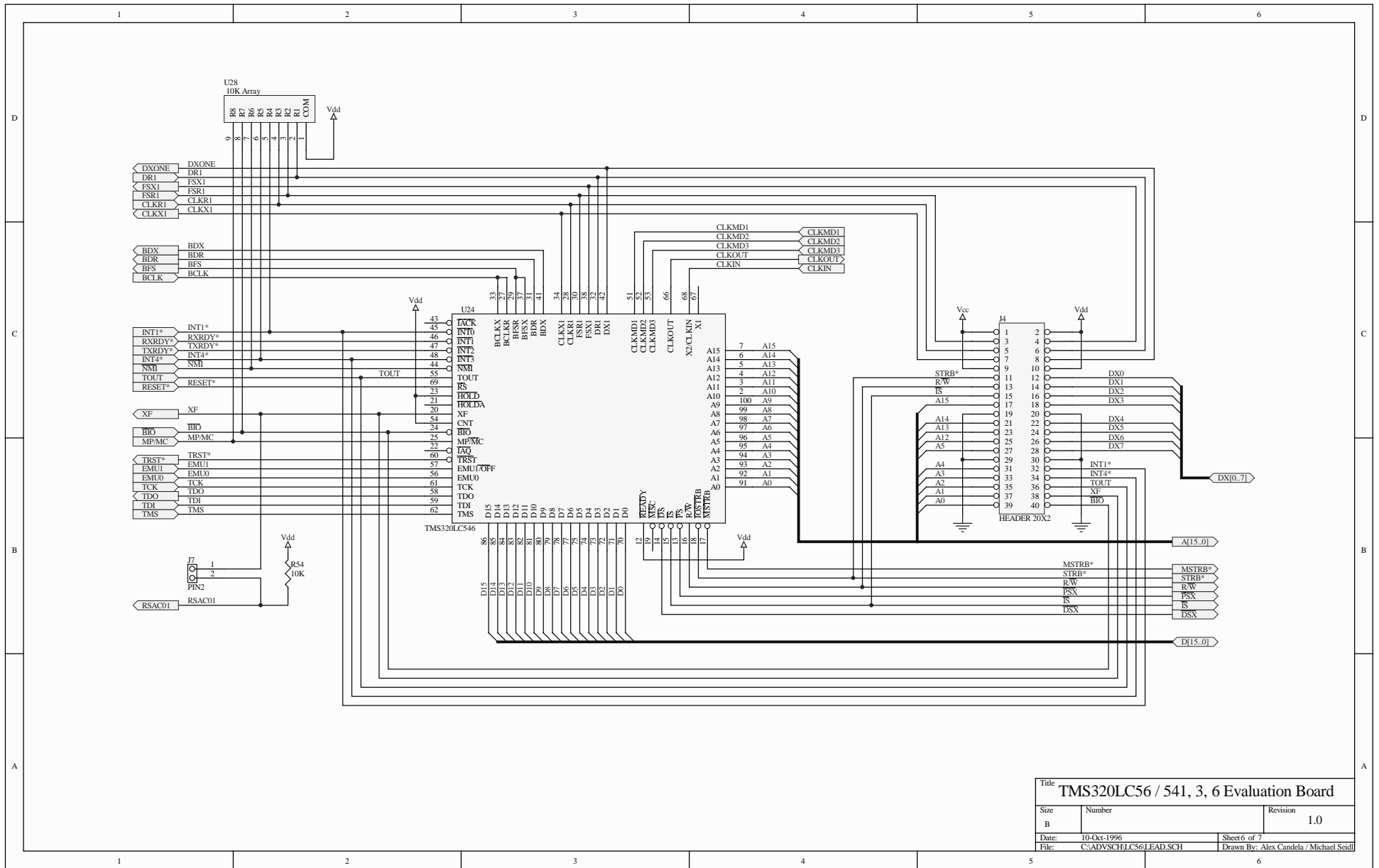


Title		
TMS320LC56 / 541, 3, 6 Evaluation Board		
Size	Number	Revision
B		1.0
Date:	10-Oct-1996	Sheet 4 of 7
File:	C:\ADV SCH\LC56\UART.SCH	Drawn By: Alex Candela / Michael Seidl

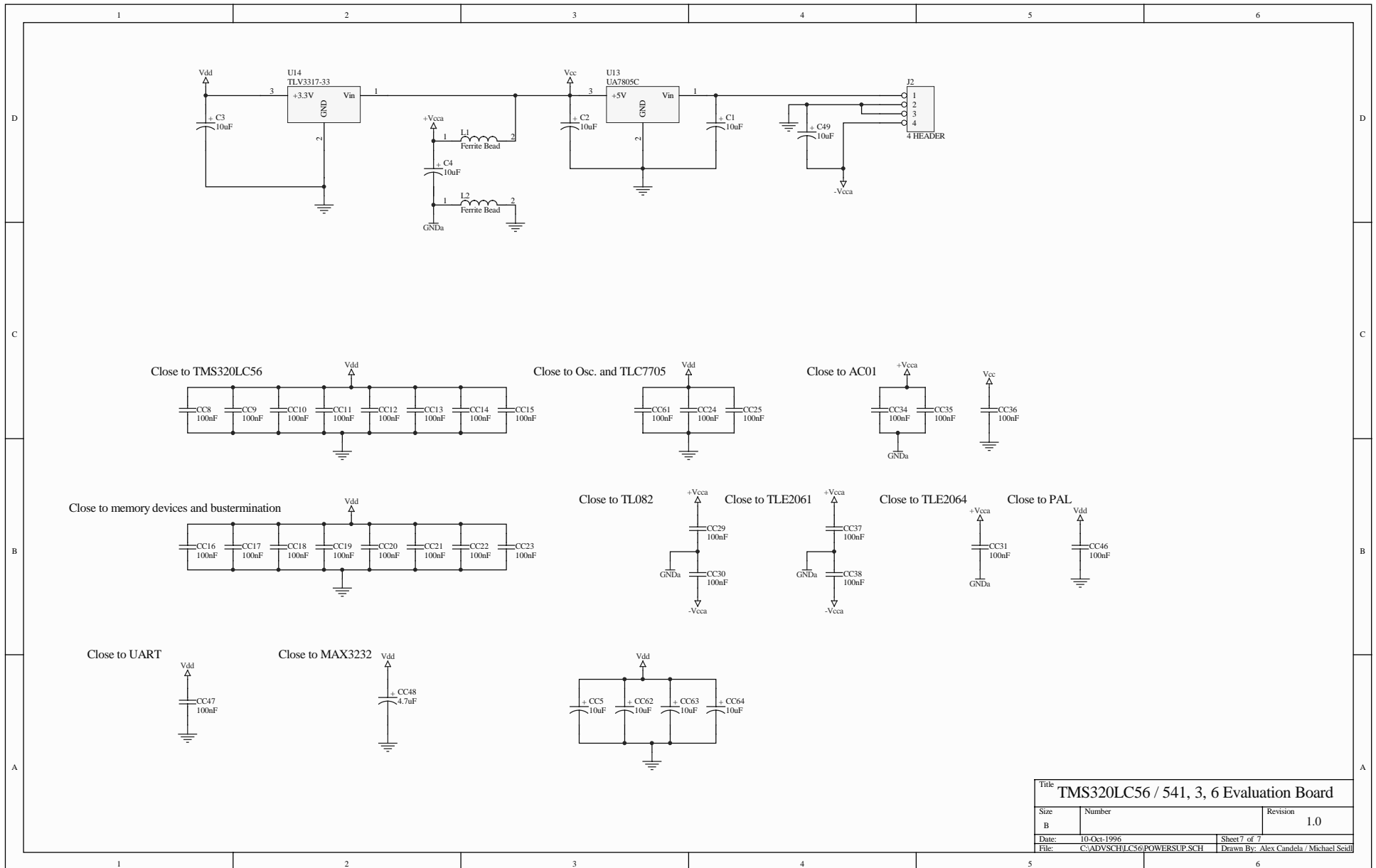


Title			TMS320LC56 / 541, 3, 6 Evaluation Board
Size	Number	Revision	
B		1.0	
Date:	10-Oct-1996	Sheet 5 of 7	
File:	C:\ADV\SCH\LC56\LINEPATH.SCH	Drawn By: Alex Candela / Michael Seidl	





Title			TMS320LC56 / 541, 3, 6 Evaluation Board
Size	Number	Revision	
B		1.0	
Date:	10-Oct-1996	Sheet 6 of 7	
File:	C:\ADVSHLC56\LEAD.SCH	Drawn By: Alex Candela / Michael Seidl	



Title		
TMS320LC56 / 541, 3, 6 Evaluation Board		
Size	Number	Revision
B		1.0
Date:	10-Oct-1996	Sheet 7 of 7
File:	C:\ADVSCH\LC56\POWERSUP.SCH	Drawn By: Alex Candela / Michael Seidl