Stony Brook University

Design Document
of
VALI-VI381 Thermal Monitoring System

Ilia Vardishvili
Zhaoqi Li
ESE 381 Embedded Design II

Table of Contents

Chapter 1: Introduction

Introduction:

VALI-381ZL Thermal Monitor System is an application, which allows monitoring the temperature of various locations in an electric vehicle. The application will continuously collect data from pre-installed temperature sensors and display the data in the screen. User can also set up the temperature limit for each sensor channel and receive warning message when the temperature exceeds the limit in a particular channel. The combination of temperature display and temperature alert provides an effective way to monitor every temperature channel throughout the vehicle. Moreover, the device features high precision temperature data from a low -cost embedded system design.

This design document contains detailed hardware and software overview. The hardware section contained descriptive design and components used in the system. The software section contains detailed overview of the programming portion the system. There are two appendixes A and B. Appendix A shows schematics of all the hardware, while appendix B lists all the source codes.

Chapter 2: Hardware

2.0 Atmega 182A

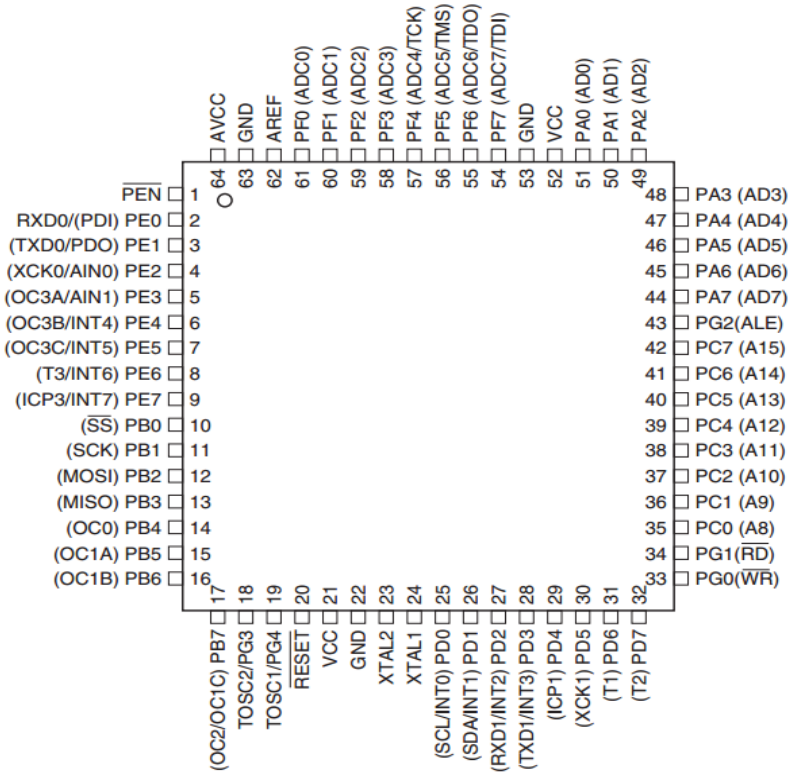


Figure 1: Pin layout of Atmel®AVR®ATmega128A . Image Curtesy of Atmel AVR 128A Data Sheet

The Microcontroller used in the system is the Atmega 128A from Atmel AVR. The High performance 8 bit Microcontroller with 128K bytes in-system flash has low power consumption that meets the requirements of the system. The chip used is as seen in figure 1 is in surface mount technology. 128A has 53 programmable I/O ports and 64-lead TQFT and pad QFN/MLF. Conventional CISC microcontroller are inferior to 128A were its 32 general purpose register can be used in a single instruction , allowing faster computations and overall speed of the system. 128A is programmed in Em bedded C, a higher level language using accommodating JTAG interface .The Key features of Atmega 128A are highlighted below. (Atmega 128A Data Sheet).

Key Feature of Atmega 182A

- Advance R8-Channel, 10-Bit ADC
- 32 General Purpose Registers, 8 bit wide
- ISC Architecture
- Maximum operating frequency of 16MHz
- Maximum External memory Space of 64kbytes
- JTAG Interface
- Two 8-bit Timer/Counters
- Two 16-bit Timer/Counters
- Dual programmable Serial USARTs
- Master/Slave SPI Interface
- Operating Voltages From 2.7V -5.5V
- 4Kbytes of EEPROM and Internal SRAM
- External and Internal Interrupt Services
- Harvard Architecture for Maximum Architecture
- Mixing of Assembly and Embedded C
- Reduced Instruction Set Computing Architecture.

* *Curtesy of Atmega 128A Date Sheet*

Atmega 128A

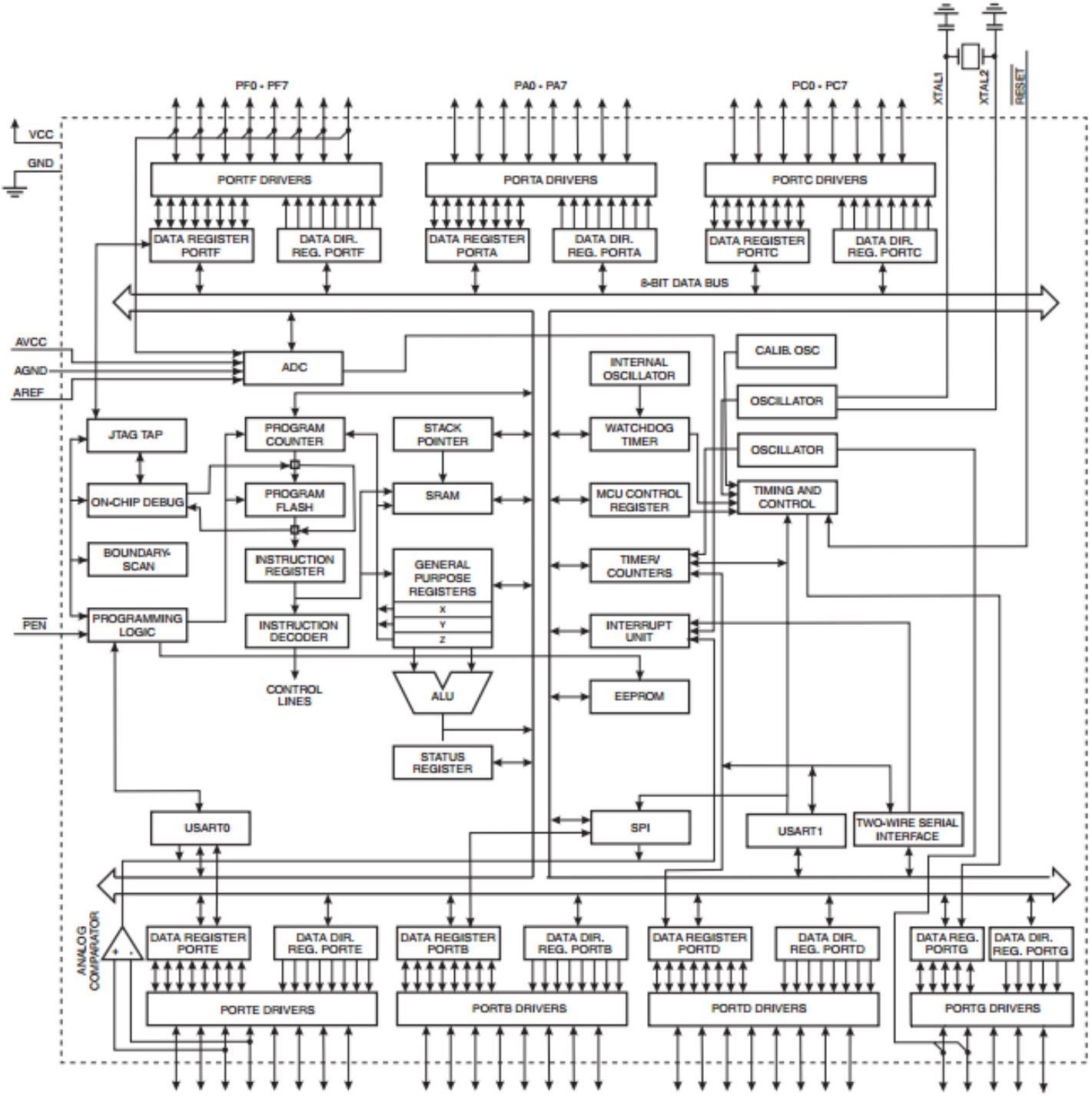
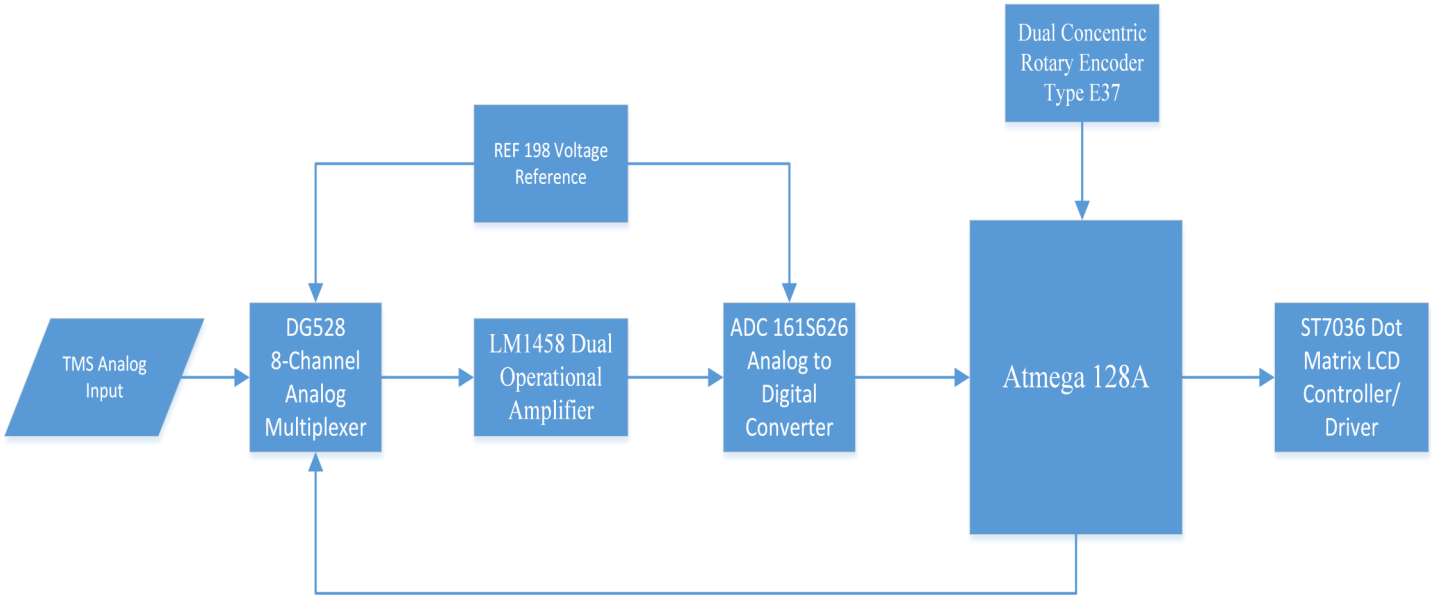


Figure 2: Block Diagram of Atmega 128A. Shown are connections of all ports and internal hardware. Curtesy of Atmega 128A Date Sheet.

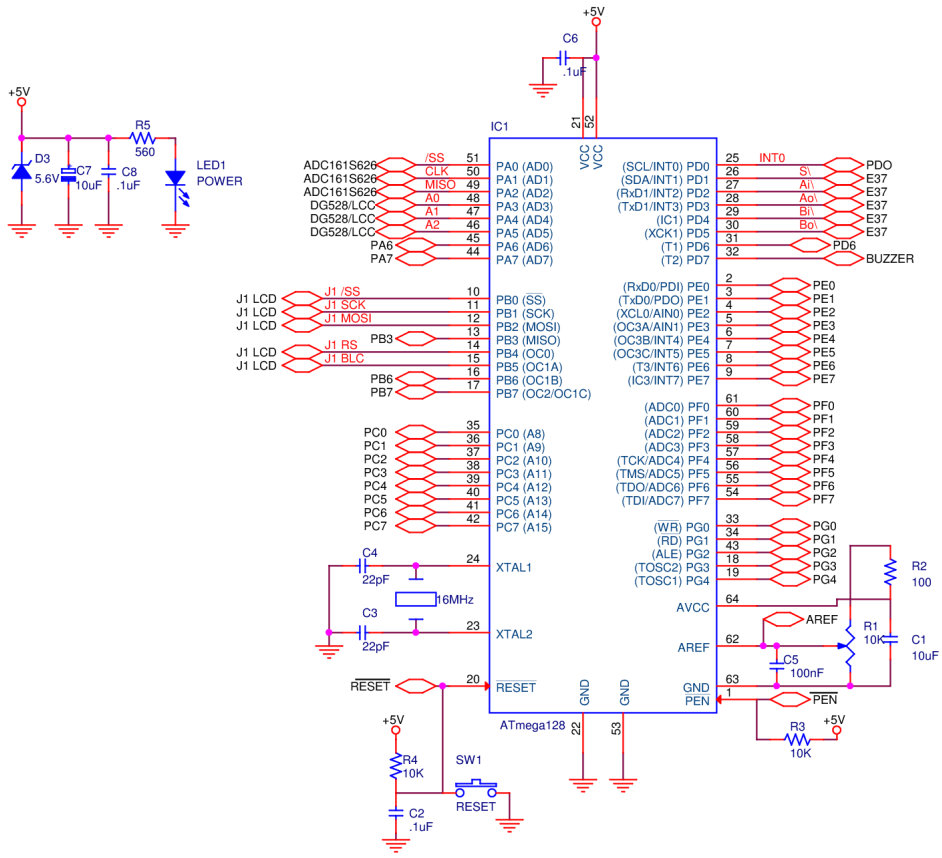
2.1 Temperature Monitoring System Hardware



Flow Chart 1: Block diagram of the TMS hardware showing that flow of data. *@ Copyright of VALI Inc.*

Hardware Description:	
Part Name	**Description/Function**
Atmega 128A	8-bit microcontroller, Governs the embedded system operations.
TMS Analog Input	Analog temperature inputs from multiple Temperature Sensors, up to 8 sensors max.
DG 528	8-channel multiplexer, selects appropriate temperature Sensors in Manual mode, or Cycle though in Auto Mode
LM 1458	Unity gain buffer, Provides high impedance input to the ADC to minimize error in conversion.
ADC 161S626	Analog to Digital Converter, converts analog temperature readings from the multiplexer to digital representation.
REF 198	Voltage reference, provides a stable voltage to the multiplexer and ADC Resulting in a ratio metric system.
E37 Rotary Encoder	Rotary encoder knob located on the from panel, provides user control of the TMS, located on the front panel
ST7036 Dot Matrix LCD	LCD Display, displays the TMS user interface, located on the front panel

2.2 Atmega 128A In System Implementation



The connections of atmega 128A in the Temperature Monitoring System can are noted in Orcad capture schematic 1. The operating voltage is 5 volts throughout the entire system. Port B of atmega 128A is designated to the LCD display screen. A JTAG connector is utilized to serially communicate with the LCD(PB0-PB2) and control backlight (BLC). Detailed description of the connection can be found in the LCD and JTAG1 portions of this section. Port A3-A5 are the control bits for DG628 multiplexer.

Schematic 1: Atmega 128A Circuit wiring of the TMS. *@ Copyright of VALI Inc.*

Atmega 128A Port Configurations	
Port#_Bits	**Description**
PortA_0-2	Used to Receive data from the ADC, the Clock (CLK) is configure as input since the ADC has internal clocking, Master In Slave Out (MISO) is configured as output to receive the binary data. Slave Select (SS) is configured as an output which controls the beginning and end of data transmission.
PortA_3-5	Used to control DG528 Channels. Configure as an output, three bits, A0-A2, are sent to the associated pins of the multiplexer to select on which channel the conversion will be administered and displays on the LCD
Port B	This port is reserved for the LCD display. A JTAG connector is used to send data to the ST7036 Dot Matrix LCD controller/driver. Port B is configured as an output providing clock signal (SCK) and /SS Slave select to begin and end transmission. Master Out Slave In (MOSI) sends the data and controlled functions. Register Select (RS) is written low due to data fact that data is only written to the LCD. No Reading from LCD is necessary. Back light control (BLC) controls the brightness level of the LCD.

Atmega 128A Port Configurations Continued

Port#_Bits	Description
Port C	Not utilized by the system.
PortD_0-5	These port pins are reserved for the E37 Rotary Encoder and configured as inputs. INT0 is utilized as a hardware interrupt that detects any action on the encoder. Ao and B0 are clock-wise and counter clockwise direction the outer knob can spin. Ai and Bi are the same direc-tion the inner knob can spin. S is the center pushbutton. Once the interrupt can be triggers by any action by the user on the E37. Once the interrupt is detected, then the software detect which of the 5 possible actions were performed.
PortD_6-7	Not utilized by the system.
PortE_0-1	Not utilized by the system.
PortE_2-7	Not utilized by the system.
PORT F	
PORT G	Not utilized by the system.

Atmega 128A Other Pin Configurations

Pin Name	Description
XTAL1 XTAL2	XTAL1 is the input to the inverting Oscillator amplifier and XTAL2 is the output. Atmega 128A has multiple clock generation options. For this instance an External Low–frequency crystal was chosen shown between the output and input.. The Crystal is 32.768 kHz clock source that configure for the 16MHz operating frequency. The value of Capacitors are de-rived from Atmega 128A Data Sheet.
AVCC	Is the Supply voltage pin for the ADC and PORT F. Atmega 128A data sheet recommends this pin to be connected to VCC at all time even if the ADC is not utilized.. If the internal ADC is in use, a low pass filter should be connected in series. The wiring configuration is based on recommended setting from the data sheet.
AREF	AREF is the reference voltage pin for the internal ADC. Atmega 128A has options of select-ing the internal reference voltage of 2.56 volts or external voltage reference using the AREF pin. The circuitry depicted in the schematic follows the recommended setup from the data sheet.
RESET	For system reset a pushbutton is implemented. When the push button is not pressed reset is high, thus system being in normal operation mode. When the pushbutton is pressed, reset is sent to low, having atmega 128A refresh.
PEN	Programming Enable (PEN) pin controls the programing of the 128A. It is internally pulled high . When it is pulled down, the system is put in SPI serial programming mode. PEN has no function during regular operating mode.

2.3 Analog to Digital Converter

Connection Diagram

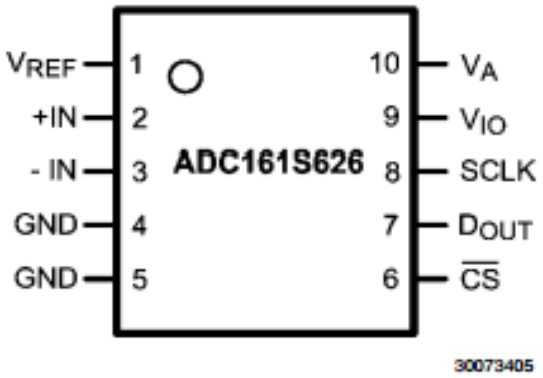


Figure 3: ADC161S626 Connection Diagram. *Curtesy of ADC161S626 Data sheet*

VA can range from +4.5V to +5.5V and VIO can range from +2.7V to +5.5V. This allows a system designer to maximize performance and minimize power consumption by operating the analog portion of the ADC at a VA of +5V while interfacing with a +3.3V controller. The serial data output is binary 2's complement and is SPI™ compatible. Information curtesy of National Semiconductor ADC161S626 data sheet.

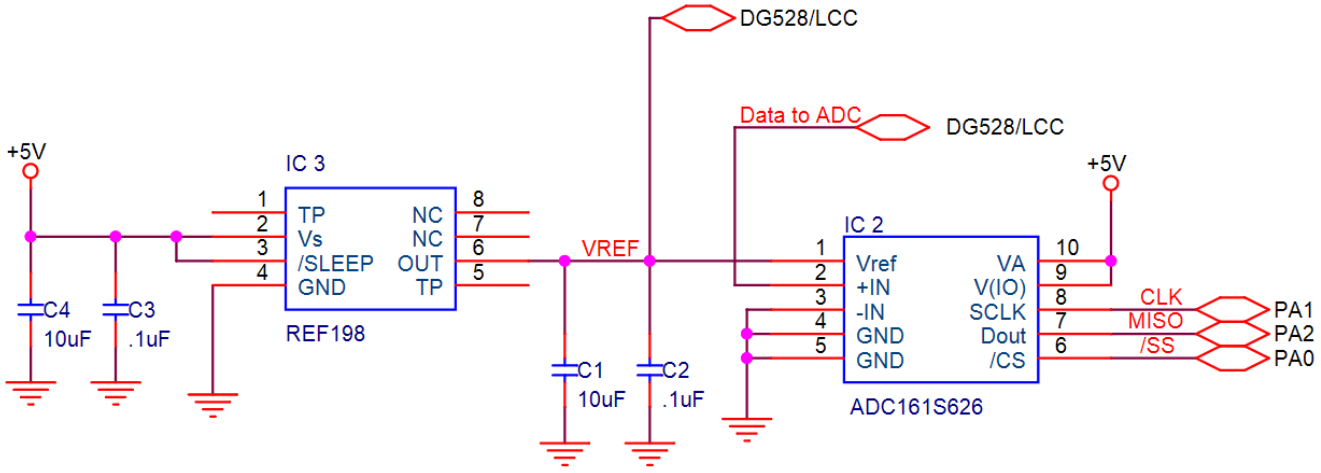
ADC161S626 Specifications

The ADC161S626 is a 16-bit successive-approximation register (SAR) Analog-to-Digital converter (ADC) with a maximum sampling rate of 250 kSPS. The ADC161S626 has a minimum signal span accuracy of ± 0.003% over the temperate range of −40°C to +85°C. The converter features a differential analog input with an excellent common-mode signal rejection ratio of 85 dB, making the ADC161S626 suitable for noisy environments. The ADC161S626

Pin Description

Pin No.	Symbol	Description
1	V_{REF}	Voltage Reference +0.5V < V_{REF} < V_A
2	+IN	Non-Inverting Input
3	–IN	Inverting Input
4	GND	Ground
5	GND	Ground
6	$\overline{CS}$	Chip Select Bar
7	D_{OUT}	Serial Data Output
8	SCLK	Serial Clock
9	V_{IO}	Digital Input/Output Power +2.7V < V_{REF} < +5.5V
10	V_A	Analog Power +4.5V < V_{REF} < +5.5V

Figure 4: ADC161S626 Connection Diagram. *Curtesy of ADC161S626 Data sheet*



Schematic 2: Wiring of ADC161S626 to atmega 128A, REF198, and DG528 multiplexer. *@ Copyright of VALI Inc.*

operates with a single analog supply (VA) and a separate digital input/output (VIO) supply.

ADC 161S626 Wiring Description

Pins 6 through 8 show the clock slave select inputs to the ADC and MISO data output to Port A 0-2 of Atmega128A. Digital I/O and Analog Power pins are put to the common power supply of 5 V. the inverting input and both grounds are grounded. Non-inverting input is connected to the output of the DG525 multiplexer. The analog input is the selected temperature sensor that is then converted to digital. The voltage reference is connected to REF198 that supplies a steady voltage of 4.096 V. Steady reference voltage is essential for precision of the data. The Reference voltage is also utilized by the multiplexer to make ratio metric system. A parallel connection is shown by the junction as the VREF bus.

2.4 REF 198: Voltage Reference

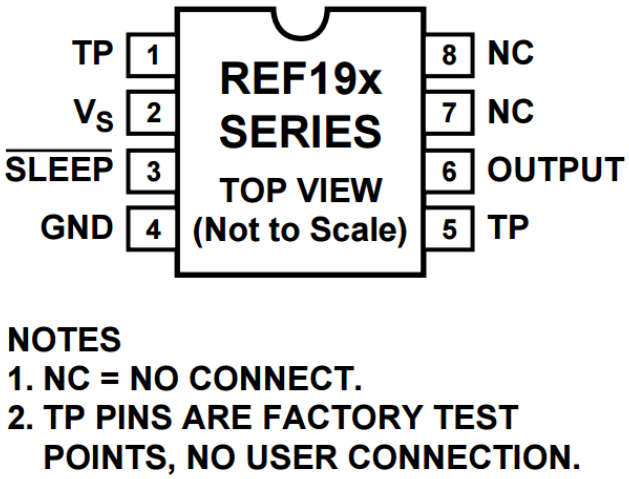


Figure 5: Ref 198 Connection Diagram. *Curtesy of REF 198 Data Sheet.*

REF 198 Specification

The REF19x series precision band gap voltage references use a patented temperature drift curvature correction circuit and laser trimming of highly stable, thin-film resistors to achieve a very low temperature coefficient and high initial accuracy. The REF19x series is made up of micropower, low dropout voltage (LDV) devices, providing stable output voltage from supplies as low as 100 mV above the output voltage and consuming less than 45 µA of supply current. *Curtesy of REF 198 Data Sheet.*

REF 198 Wiring Description

The voltage source (Vs) and sleep pins of the IC are connected to common supply voltage of 5 V. The sleep mode is initiated when pin is low, having it high calls for continues operation.. Two decoupling capacitors are connected between the voltage source and Vs pin for nominal operation (REF198 Data Sheet). Ground pin is connected to common ground. NC and TP pins are not connected as noted in Figure 5. The output voltage is 4.096 for REF 198 series as stated in its data sheet. The output provides reference voltage for the ADC and the multiplexer.

2.5 DG 528/LLC 8-Channel Analog Multiplexer

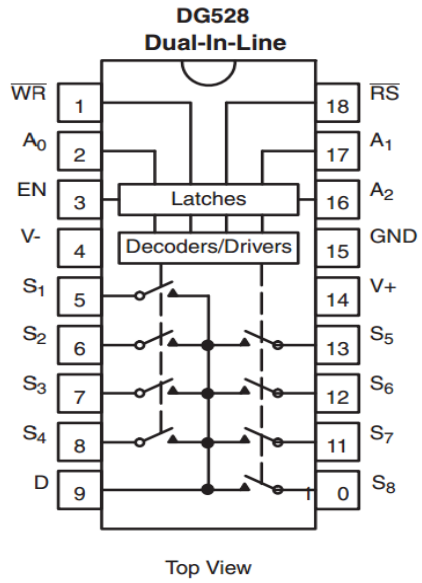


Figure 6: DG Pin Configuration. *Curtesy of DG 528 Data Sheet* .

General Description

The DG528 is an 8-channel single -ended analog multiplexer de-signed to connect one of eight inputs to a common output as de-termined by a 3-bit binary address (A0 , A1 , A2).

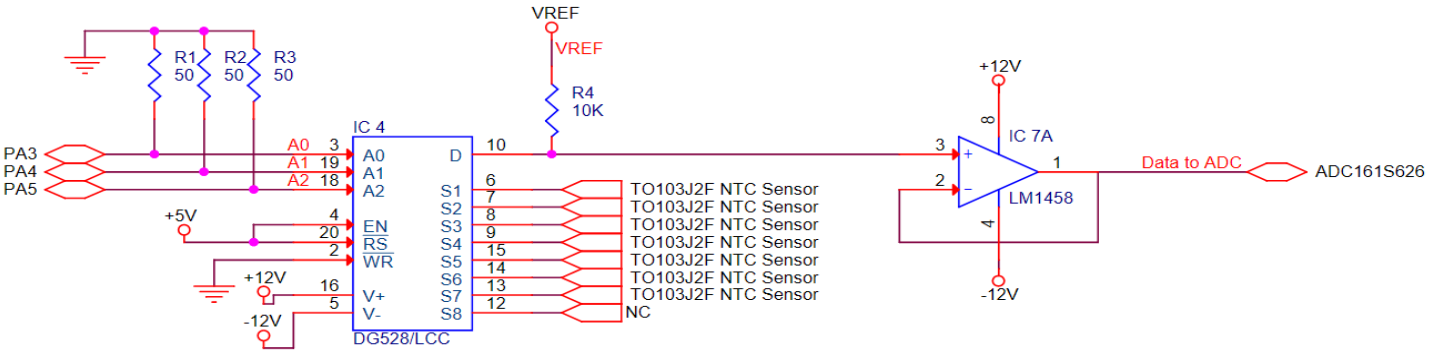
FEATURES

- Low RDS(on): 270 :
- 44 V Power Supply Rating
- On-Board Address Latches
- Break-Before-Make
- Low Leakage - ID(on): 30pA

Curtesy of G 528 Data Sheet .

TRUTH TABLE - DG528						
8-Channel Single-Ended Multiplexer						
A_2	A_1	A_0	EN	WR	RS	On Switch
Latching						
X	X	X	X	↓	1	Maintains previous switch condition
Reset						
X	X	X	X	X	0	None (latches cleared)
Transparent Operation						
X	X	X	0	0	1	None
0	0	0	1	0	1	1
0	0	1	1	0	1	2
0	1	0	1	0	1	3
0	1	1	1	0	1	4
1	0	0	1	0	1	5
1	0	1	1	0	1	6
1	1	0	1	0	1	7
1	1	1	1	0	1	8

Figure 7: Truth table for selection of channels containing the temperature sensors. *Curtesy of G 528 Data Sheet* .



Schematic 3: Wiring of the DG528 multiplexer (left) and LM1485 Dual Operational Amplifier in the System. *@ Copyright of VALI Inc.*

DG 528 Wiring Description	
Pin Name	**Description**
A0-A2	This pins are the input pin driven by atmega128A, which select which temperature channel to latch.
Enable (EN)	Enable pin allows the latching of the multiplexer. When set high it allows the system to continu-ously to be able to select a channel.
RS/	This pin is set high which stores the previous data of selected latch, otherwise the multiplexer might unlatch.
Write(WR/)	This pin controlles whether the DG 528 is being written. The write pin must be grounded as specified by the data sheet, in order to latch the chip at any time.
S1 –S8	This pins are where the seven analog temperature sensors are connected to. S8 is not connected.
D	D is the output of the multiplexer. The selected channel is outputted here in form or a resistance since the NTC sensors are essentially variable resistors. The 10K Ohms pull up resistor acts as a voltage divider that converts the temperature into associated voltage. The pull up voltage VREF is connected from the REF198 as mentioned in REF198 section. The voltage is then passed into the unity gain buffer, LM1458, and then to the ADC.

2.6 LM1458 Dual Operational Amplifier

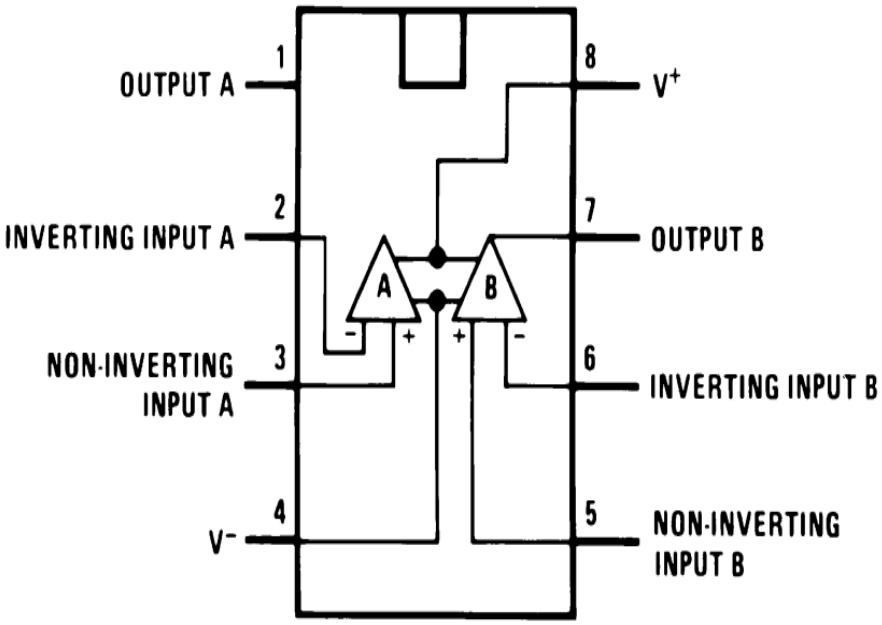


Figure 8: LM 1458 Dual operational Amplifier Pin Layout. *Curtesy of LM1458 Data Sheet* .

General Description

The LM1458 and the LM1558 are general purpose dual operational amplifiers. The two amplifiers share a common bias network and power supply leads. Otherwise, their operation is completely independent.

Features

- No Frequency Compensation Required
- Low-Power Consumption
- Short-Circuit Protection
- 8-Lead TO-99 and 8-Lead PDIP
- No Latch Up When Input Common Mode Ranges Range is Exceeded

LM 1458 Wiring Description

Pin Name	Description
Non-Inverting	The analog output voltage of DG528 multiplexer is connected to this pin. Non inverting op-
Inverting Input	Inverting input is connected to the output of the amplifier. This configures the amplifier as a
Output A	Is the output voltage from the non-inverting input with high impedance. The filtered voltage goes into the analog to digital converter.
V+, V-	The operational power rating for the LM1485 IC is ±12V. V+ is connected to +12V and V– is connected to –12V.
Inputs of Op-amp B	Not connected.

2.7 Dual Concentric Encoder Type E37

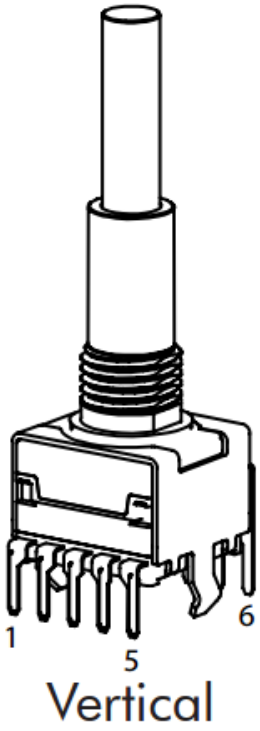


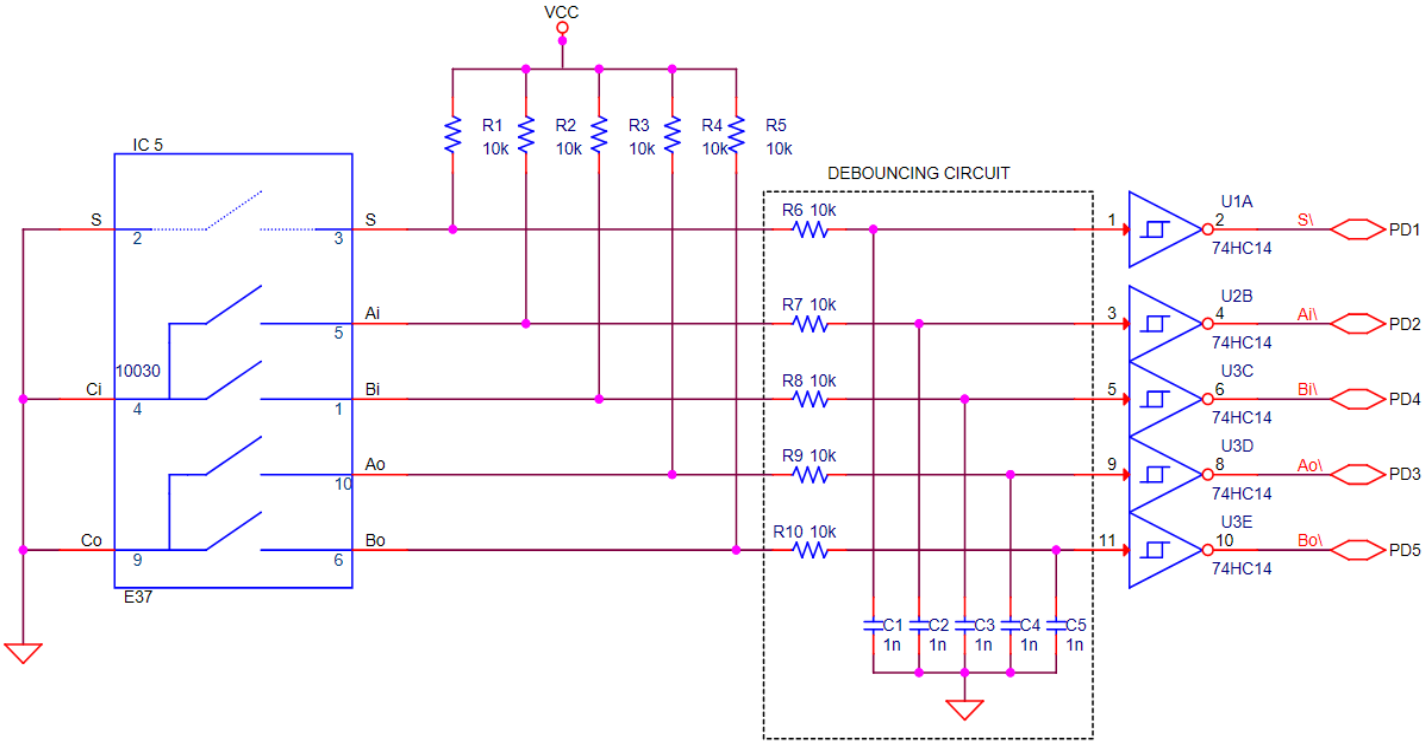
Figure 9: E37 Pin Layout. *Curtesy of E37 Data Sheet* .

Pin#	1	2	3	4	5
	Bi	S	S	Ci	Ai
	6	7	8	9	10
	Bo	NC	NC	Co	Ao

Figure 10: E37 Pin Assignment. *Curtesy of E37 Data Sheet* .

Key Feature:

- Dual concentric: Two encoders - one space
- 16 or 32 detents standard resolution
- With or without integrated push button
- Rotational life: Up to 1,000,000 revolutions
- Excellent indexing feel with 0.5, 1.0, 1.5, 2.0 or 2.5
- Ncm switching torque (remains consistent over life)
- Gold plated contacts
- Robust metal housing with metal shaft 11.5 x 12.3 x 9.1 mm body size
- Optional IP68 front panel sealing
- Operating temperature range: -40 to +85°C



Schematic 4: Wiring of Dual Concentric Encoder Type E37 to the Atmega 128A. *Curtesy of E37 Data Sheet* .

2.8 Dot Matrix LCD Controller/Driver

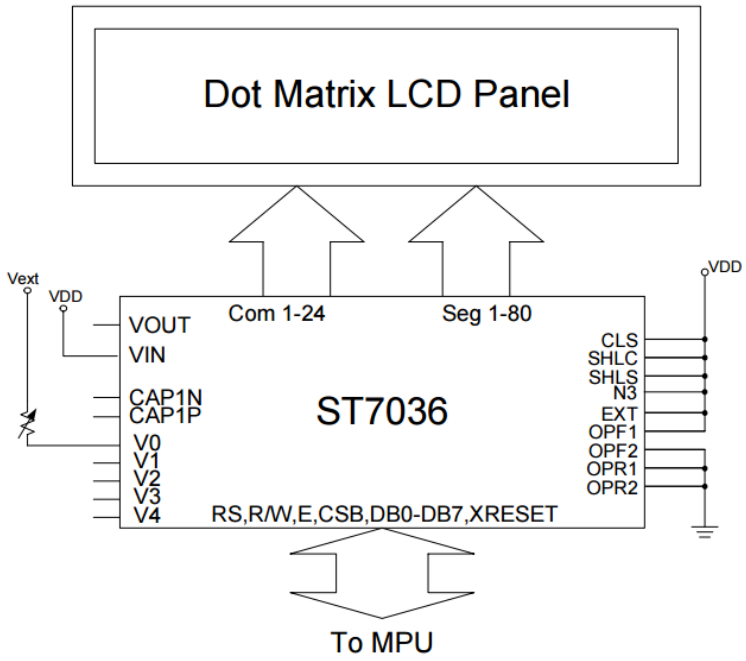
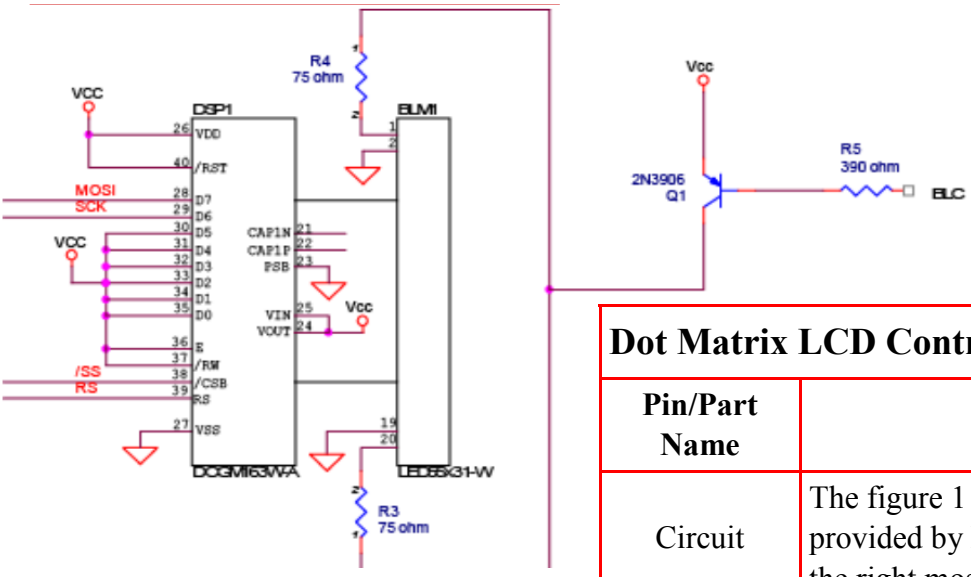


Figure 11: Dot Matrix LCD Controller/Driver Flow Diagram. *Curtesy of Sitronix ST7036 Data Sheet.*

Features:

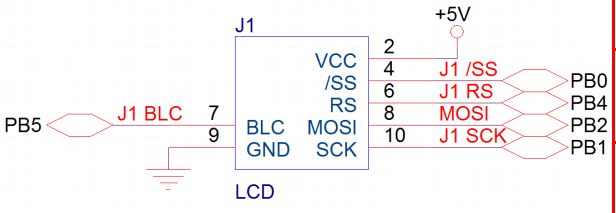
- 5 x 8 dot matrix possible
- Low power operation support: -- 2.7 to 5.5V
- Range of LCD driver power -- 2.7 to 7.0V z 4-bit, 8-bit, serial or 400kbits/s fast I2 C-bus MPU interface enabled
- 80 x 8-bit display RAM (80 characters max.)
- 10,240-bit character generator ROM for a total of 256 character fonts(max)
- 64 x 8-bit character generator RAM (max)
- Support two display mode: 16-com x 100-seg and 80 ICON 24-com x 80-seg and 80 ICON
- 16 x 5 –bit ICON RAM(max)

Curtesy of Sitronix ST7036 Data Sheet.



Schematic 5: SPI and Wiring connections of the DOG module. @ Copyright Kenneth Short , 2006.

The ne Aliases shown in figure are connected to Port B of atmega 128A using a JTAG connector shown



Schematic 6: JTAG Connecter linking LCD to Atmega 128A. *@ Copyright of VALI Inc.*

Dot Matrix LCD Controller/Driver Description	
Pin/Part Name	**Description**
Circuit	The figure 1 shown, is a side board built and provided by Kenneth Short. The LCD rests in the right most panel displayed.
MOSI(D7)	The SPI data input from Atmega 128A
SCK	Clocking delivered by128A
/SS	Slave select for SPI data transfer, controlled by Atmega 128A
RS	Register select, starts and ends transfer of data, controlled by Atmega 128A
BLC	Black light control settings, set to full scale, but brightness can be adjusted, controlled by Atmega 128A.

Dual Concentric Encoder Type E37 Wiring Description

Pin/Part Name	Description
Circuit	The E37 encoder outputs are connected to resistor pull-up network, which then is processed though a denouncing circuit, shown in the dotted are of schematic 4. Finally, it is filtered through inverting Schmidt triggers before connecting to atmega 128A. The setup of the circuit is the recommended configuration from the E37 Data sheet.
S(3)	S is the output of the center push button utilized by the system.
Ai, Bi	These are the outputs of the inner nob rotations. The rotations are either clockwise (A) or counter clockwise).
Ao, Bo	These are the outputs of the outer nob rotations. The rotations are either clockwise (A) or counter clockwise).
S(2), Ci, Co	Are all grounded, its functions are not specified in the E37 data sheet
R1-R5	Pull up resistor network, values and setup specified by the data sheet.
De-bouncing Circuit	Highlighted by dash line in schematic 4, is the de-bouncing circuit for the rotary encoder. R6 through R10 are 10KOhms, C1-C5 are 1nF. The circuit is designed after recommended configuration by the E37 data sheet.

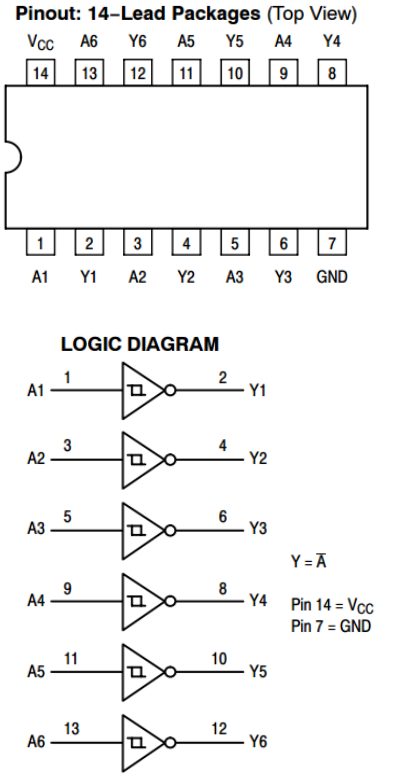


Figure 12: 74HC14 Pin Assignment (Top) and Logic Diagram (Bottom). *Curtesy of 74HC14 Data Sheet .*

Features:

- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS and TTL
- Operating Voltage Range: 2.0 to 6.0 V
- Low Input Current: 1.0 A
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance With the JEDEC Standard No. 7A Requirements
- ESD Performance: HBM 2000 V; Machine Model 200 V
- Chip Complexity: 60 FETs or 15 Equivalent Gates
- These are Pb−Free Devices

Curtesy of 74HC14 Data Sheet .

FUNCTION TABLE

Inputs	Outputs
A	Y
L	H
H	L

Tables 1: Function table of 74HC14. *Curtesy of 74HC14 Data Sheet .*

2.9 TO103J2F NTC Sensor

Figure 13: TO103J2F NTC Sensor physical. *Image curtesy of Kenneth Short & U.S. Senor CORP.*

Specifications:

Resistance @ +25°C = 10,000 Ω ± 10%

Resistance/ Temperature Curve = "J"

Beta "β" (0 to +50°C) = 3,892 K Nominal

Temperature Coefficient @ +25°C = -4.4 %/°C Nominal

Maximum Temperature Rating = +150°C

Curtesy of U.S. Senor CORP. TO103J2F NTC Sensor Data Sheet.

Description:

These sensors will be connected to the inputs of the DG528 multiplexer. The sensors can be place at user defined locations.

2.9.1 Alert Buzzer

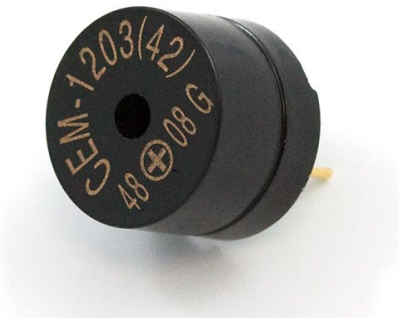


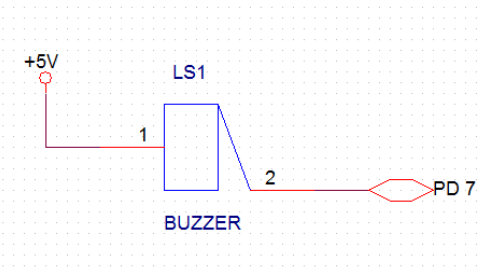
Figure 14: Piezo Speaker 12mm 2.048kHz, *Curtesy of SparkFun Electronics inc.*

Specifications:

This is a small 12mm round speaker that operates around the audible 2kHz range. You can use these speakers to create simple music or user interfaces.

Each speaker is PTH solderable and requires an operating voltage of 3.5-5V with a mean current of 35mA max. These speakers also have a typical sound output of 95 dBA and a coil resistance of 42 ±6.3 ohms.

Curtesy of SparkFun Electronics & CUI Inc. Piezo speaker Data Sheet.



Schematic 7: Connections of the Alarm buzzer to the Atmega 128A. *@ Copyright of VALI Inc.*

Description:

When the temperature exceeds the user defined limit, a built in buzzer will sounds the alarm. Press of the center pushbutton of the E37 rotary encoder will silence the alarm for a period of time.

Chapter 3: Software

3.1 Software Overview

Program Description

This program is designed to drive a temperature sensing system which monitors 7 temperature sensors located throughout an electric car. The program will receive analog data from the selected sensor channel, then load the data into the 16-bit ADC. The Microcontroller will then receive the digital data from the ADC through a SPI interface. The Data received by the Microcontroller will then be converted to the temperature reading using a table lookup function. The system will take the temperature data and display to the users in a LCD screen. The system can also monitor the temperature and send out alert if the temperature goes above the user defined limit for each channel. The alert system includes an alert message and buzzer to make effective warnings to the users. The system is designed to be low cost and able to provide high accuracy temperature data to the user. It is ideal to be installed in an electric car with battery voltage supply. This program includes 7 major components: Main function, DG528 analog multiplexer driver, LCD driver, Finite State Machine, Table lookup, Interrupt Service Routine, Analog to Digit Converter driver. See Table of File Description for the detail of each component.

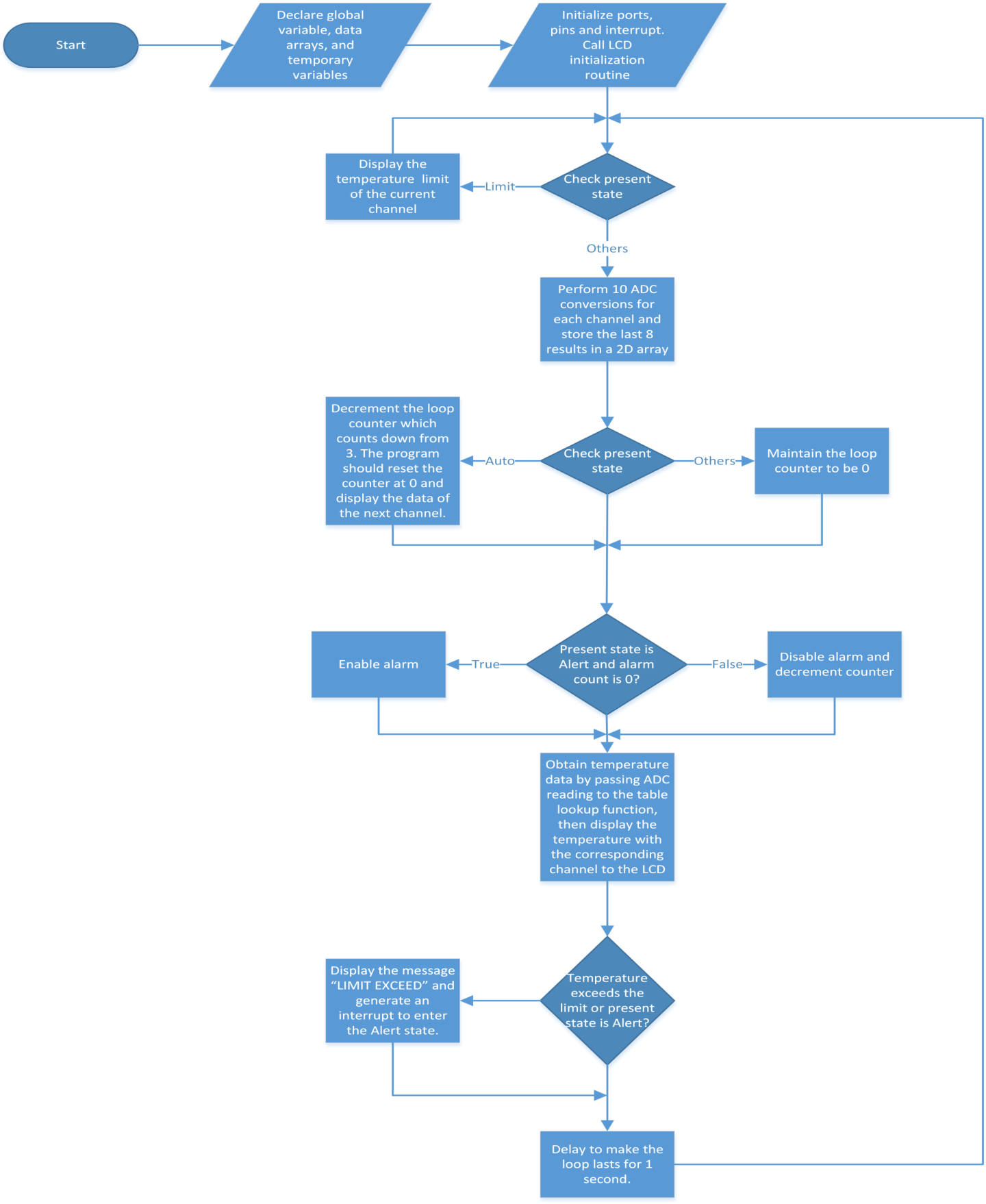
This program includes 7 major components: Main function, DG528 analog multiplexer driver, LCD driver, Finite State Machine, Table lookup, Interrupt Service Routine, Analog to Digit Converter driver. See Table of File Description for the detail of each component.

3.2 System Program Index

Name	Type	Description	Page	Included Files
Main.c	C File	C File for main function and major variables in this program.	36	<iom128.h>, <avr_macros.h>, <intrinsics.h>, <stdio.h>, "LCD_Driver.h", "DG528_driver.h", "ADC161S626_driver.h","fsm.h"
fsm.h	H File	Header file for the Finite State Machine typedef variables and structs.	40	N/A
fsm.c	C File	C source file for the Finite State Machine function and arrays definitions.	41	"fsm.h"
fsm_fn.c	C File	C source file for the Finite State Machine task functions definition.	44	<iom128.h>, <avr_macros.h>, <intrinsics.h>
LCD_Driver.h	H File	Header File for the LCD external variables and functions templates	46	N/A
LCD_Driver.c	C File	C souces file for the LCD initialization, data writing and command writing routines.	47	<iom128.h>, <avr_macros.h>, <intrinsics.h>
LCD_ext.c	C File	C sources file for LCD display buffers manipulation routines.	52	"LCD_driver.h"
DG528_driver.h	H File	Header file for the Analog mux control routine template.	54	N/A
DG528_driver.c	C File	C source file for the Analog multiplexer channel selection routine.	55	<iom128.h>, <avr_macros.h>, <intrinsics.h>
ADC161S626_driver.h	H File	Header file for the A to D Converter function template.	56	N/A
ADC161S626_driver.c	C File	C source file for the A to D Conversion routine.	57	<iom128.h>, <avr_macros.h>
isr.c	C File	C souce file for all the Interrupt Service Routine s(ISR).	59	<iom128.h>, <avr_macros.h>, <intrinsics.h>, "fsm.h"
fcn_tb.c	C File	C souce file for the table lookup function and temperature table.	61	N/A

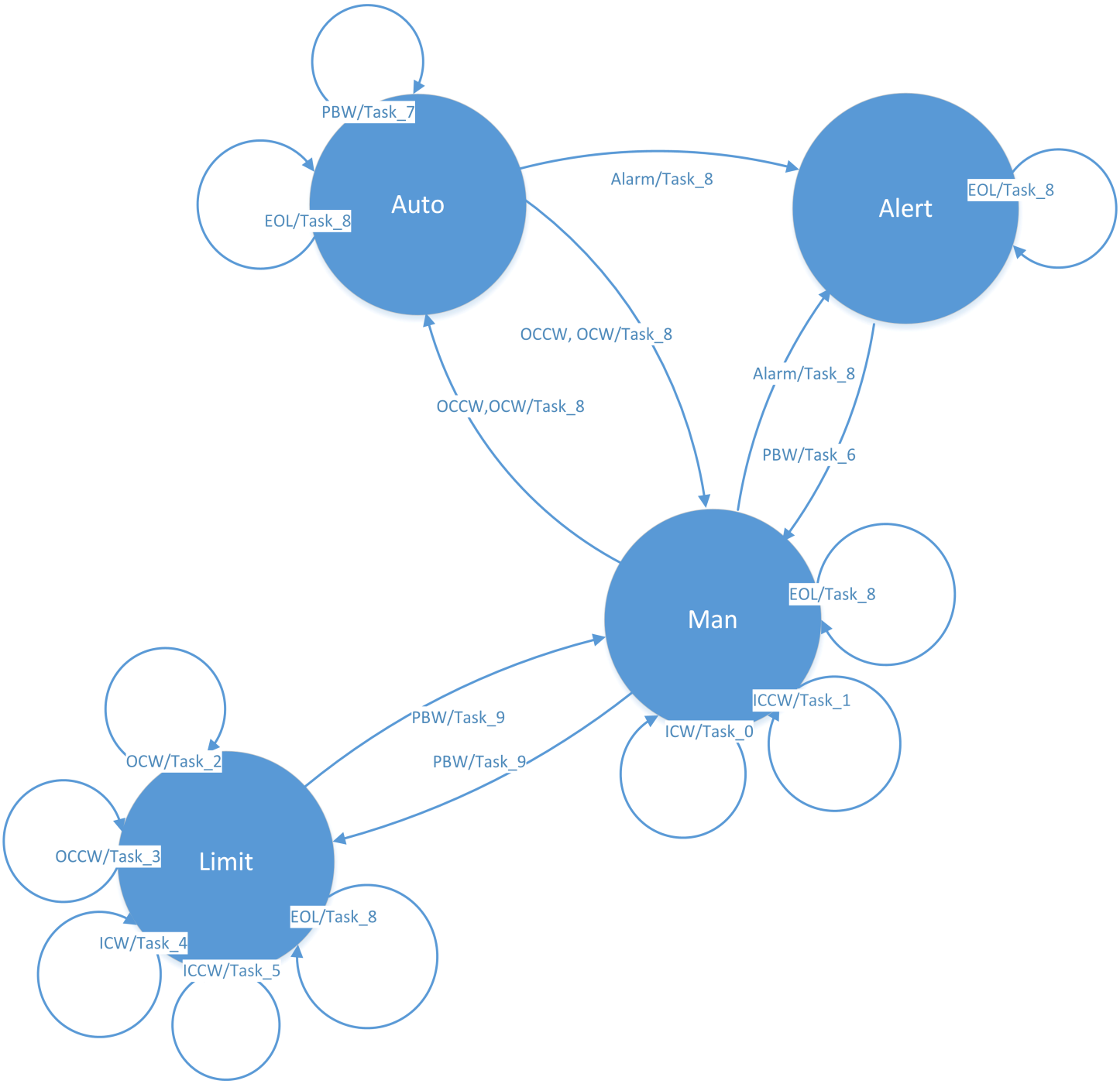
Table 2: All main program listing of the Temperature Monitoring System. Page tab lists page of source code of the programs. @ *Copyright of VALI Inc. The source files are located in Appendix B*

3.3 System Program Flow Chart



Flow Chart 2: Flow chart of the program structure. *@ Copyright of VALI Inc.*

3.4 The Finite State Machine



Flow Chart 3: Structure of the Finite State Machine, showing all the states and associated Functions. *@ Copyright of VALI Inc.*

Finite State Machine Description

State	Description	State transition/Function call
Auto	In the Auto state, the system will automatically display the temperature readings from each channel for 3 seconds. The temperature readings will be updated by the system each second. The display path is Ambient outside -> Motor -> HV Controller -> DC/DC Converter -> Battery Box 1 -> Battery Box 2 ->Ambient inside. The System will roll over at the last channel.	Auto - Auto : PBW/Task_8 ; Auto - Auto : EOL / Task_9; Auto - Man : OCCW/ Task_9; Auto - Man : OCW/ Task_9; Auto - Alert : Alarm/Task_9;
Man	In the Man state, the system will display the temperature reading from the current channel until user changes the channel using the inner knob. Incrementing the channel at Ambient inside will roll over to Ambient outside. Decrementing the channel at Ambient outside will reset to Ambient inside. When the system transfer from Auto state to the Man state, the current channel in the Auto state will become the current channel in the Man state.	Man - Auto : OCCW/Task_9; Man - Auto : OCW/Task_9; Man - Man : ICW/ Task_0 Man - Limit : PBW/Task_9; Man - Alert : Alarm/Task_9; Man - Man : EOL/Task_9;
Limit	In the Limit state, user will gain access to the temperature limit of the current channel. User can increment the temperature limit by turning the knob clockwise and decrement the temperature limit by turning the knob counter clockwise. The size of the change is 1 for inner knob and 10 for outer knob. The temperature reading can vary from 0 to 94 and the limit will simply roll over if the user inputs a number greater than 94. The temperature limits are written to the eeprom so user will be able to save their limit setting.	Limit - Man : PBW/Task_9; Limit - Limit : OCW/Task_2; Limit - Limit : OCCW/Task_3; Limit - Limit : ICW/ Task_4; Limit - Limit : ICCW/Task_5; Limit - Limit : EOL/Task_9;
Alert	When the system detects a temperature reading higher than the temperature limit for the current channel, it will enter the Alert state and display a "Limit Exceed" message to warn the user. The buzzer will also be enabled and continuously beeping until the user presses the pushbutton to enter the Man state and slient the buzzer. If the temperature is still above the limit after the press, the system will re-enter the Alert system, but the buzzer will resume to make beep after 8 seconds.	Alert - Man : PBW/Task_7; Alert - Alert : EOL/Task_9;

Task Functions of The FSM

Task	Description	Function
task_0	update the current channel to the next channel	incr_channel()
task_1	update the current channel to the previous channel	decr_channel()
task_2	Increase the temperature limit of the current channel by 10 and roll over to 0 at 94	Incr_limit_10()
task_3	Decrease the temperature limit of the current channel by 10 and reset to 94 at 0	decr_limit_10()
task_4	Increase the temperature limit of the current channel by 1 and roll over to 0 at 94	incr_limit_1()
task_5	Decrease the temperature limit of the current channel by 1 and reset to 94 at 0	decr_limit_1()
task_6	Toggle a flag to change the temperature readings between Celsius and Fahrenheit.	dsp_f()
task_7	Start a counter to slient the buzzer for 8 seconds, the buzzer will resume when the counter overflow	start_counter()
task_8	Do nothing and return	null_fn()

Table 3: This tables represents all the functions called by the finite state machine. *@ Copyright of VALI Inc.*

Interrupts of the System

Interrupt	PIN/PORT	Trigger type	Duty	Comments
INIT0	PD0	Positive Edge	Assign the key "alarm" to the FSM	Triggered by a positive pulse generated at PortD_bit0.
INIT1	PD1	Negative Level	Assign the key "pbw" to the FSM	Triggered by the pushbutton of the encoder; must check for button release before the program returns to main.
INIT2	PD2	Any Edge	Assign the key "icw" to the FSM when the program detects a different voltage levels at PIND_4 and PIND_2; assign "iccw" otherwise	Trigger when turning the inner knob of the quadrature shaft encoder by 1 click. (For details check out the hardware description of the rotary encoder)
INIT3	PD3	Any Edge	Assign the key "ocw" to the FSM when the program detects a different voltage levels at PIND_5 and PIND_3; assign "occw" otherwise	Trigger when turning the outer knob of the quadrature shaft encoder by 1 click. (For details check out the hardware description of the rotary encoder)

Table 4: List of interrupts generated by the hardware. The physical connection of the interrupts and their descriptions are shown. *@ Copyright of VALI Inc.*

Additional comments on the FSM structure

Definitions:

State: type defines 4 states in the FSM. See the table State for details.

Key: type defines the possible key input to the FSM. See the state transitions table Key for details.

Task_fn_ptr: type defines the pointer to a particular task function executed by the FSM.

Transition: a basic structure which contains Key, next state, and function pointer to the task function. Each transition will define the state transition caused by a particular input to the FSM.

Procedure:

*Each state in the FSM should have one transition structure for the corresponding input to the FSM. However, if the input neither results in any state transition nor calls a special task function, this input can be categorized as eol of the state. The eol transition must not change the current state or call any special function. In the array of transition structure for each state, there should be one exact eol transition structure located at the end of the array.

*One array of transition structure is required for each state.

*In order to select the right state transition, the FSM function must go through an array of pointer to the beginning of each transition arrays and select the proper array for the present state. In order to do so, the array of pointer must strictly follow the enumeration of the state in the type definition. For instance, if state “Auto” is place at the beginning of the state definition, the corresponding transition array should also be placed at the beginning of the pointer array. After the FSM select the correct pointer, it should select the appropriate transition based on the given key. If the key does not match, the FSM should simply select eol so that the entire system remain unchanged.

State Transitions:

The transition from one state to another is cause by the user input. The control are based on the actions on the E37 Rotary Encoder. A Key value is passed though the FSM after an interrupt has been triggered by the user controls. The list of all possible actions and their associated interrupts are as follows:

State Transitions Key Table

Key	Description	Inter-rupt	Condition
ocw	Outter knob turned clockwise by 1 click	INIT3	PD5 xor PD3 == 1
occw	Outter knob turned counter clockwise by 1 click	INIT3	PD5 xor PD3 == 0
icw	Inner knob turned clockwise by 1 click	INIT2	PD4 xor PD2 == 1
iccw	inner knob turned counter clockwise by 1 click	INIT2	PD4 xor PD2 == 0
pbw	pushbutton is pushed	INIT1	N/A
alarm	program enters the Alert state by generating a software interrupt at PB0	INIT0	A positive pulse at PD0
eol	End of line or invalid key input	N/A	program does not find a matched input to trigger the state transition of the current state

Table 5: User input keys that control the program. Key is the physical actions on the E37 rotary encoder by the user. Key generates an associated interrupt, and the interrupt changes the state of the system. *@ Copyright of VALI Inc.*

2.5 Table lookup

The ADC reading to Temperature reading conversion table is stored in the SRAM. Since the measured temperature ranged from -30 to 94 degree Celsius, there will be 124 entries in this table. Instead of implementing an array of structures, this system uses for loops to search the array of ADC readings until it finds the first element that is smaller than the given ADC reading. The program will then track down the index of the element, the value of the element, and the value of the previous element. If no element is matched, it indicates that the temperature is higher than the measurement range. The program will simply return 940 (explained later) to the main function. If a matched element is found and the required values have been recorded, the program will employ the method of linear interpolation to approximate the temperature to a precision of 0.1C. The equation is given by:

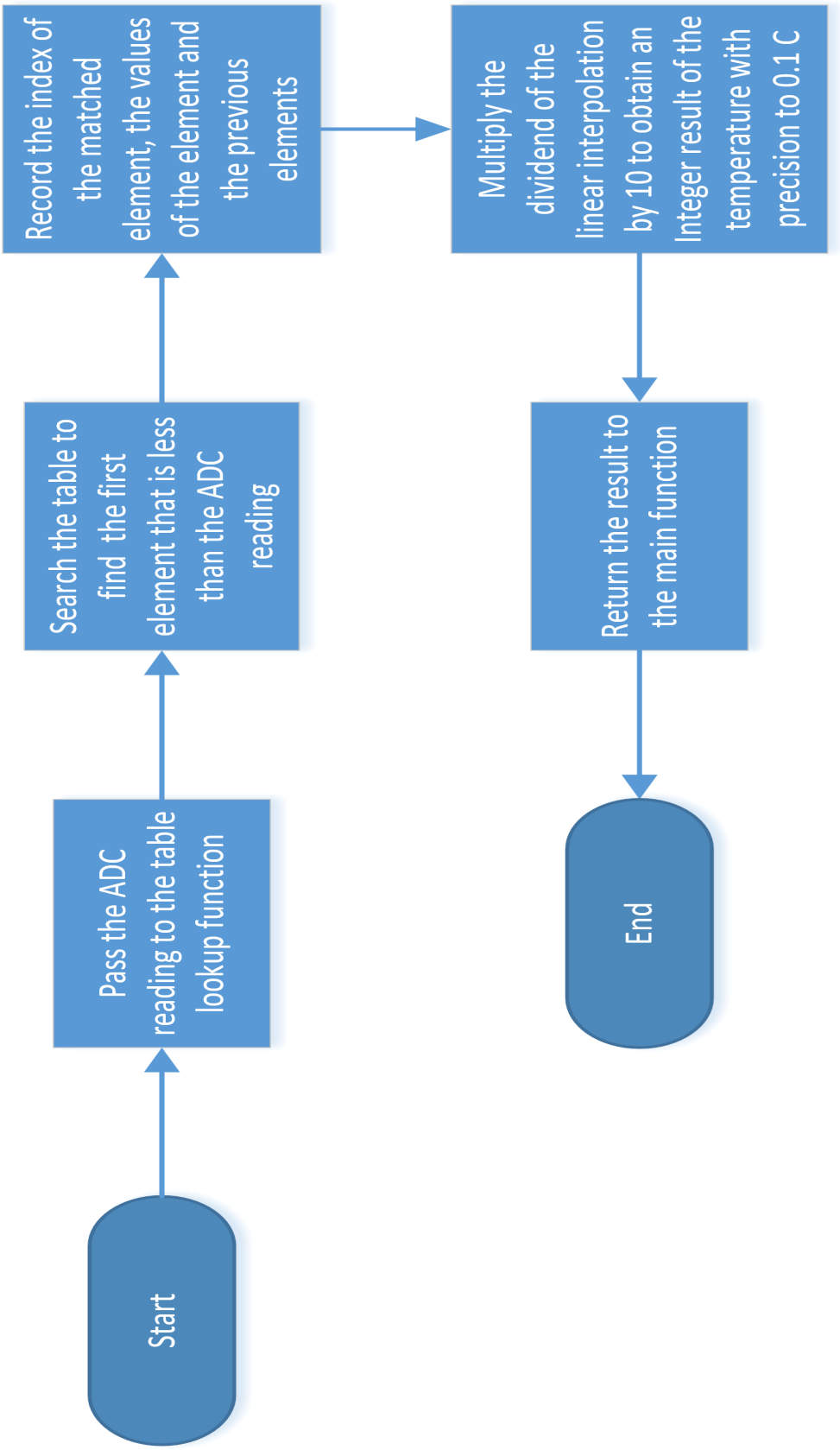
$$f(x) = temp - (result - var_b) * \frac{10}{var_a - var_b}$$

Equation 1: Function determines the temperature by linear interpolation

Where temp = (index – 30)*10 and result is the given ADC reading. var_b is the value of the element indicated by the index, var_a is the element of the previous index. The dividend of the equation need to multiply by 10 in order to obtain an Integer result with precision to 0.1C. In the main function which receives the temperature from the table lookup function, it will parse the Integer result and store the integer part and the decimal part into two different character variables. This approach eliminates the addition of the floating point routines to the program and therefore reduces the size of code needed to compile.

*Since the table is stored in SRAM, the whole 124 entries will be break into 4 small tables of 31 entries each. To obtain the correct index from these tables, simply add the numbers of elements in the previous table to the found index and use the result as the final index. For instance, if an element is found at table_2 index 15, the final index should be 31 + index.

Table Look-Up Flow Chart



Flow Chart 4: Table look up for the systems method of getting the temperature reading from the binary ADC value. *@ Copyright of VALI Inc.*

2.6 LCD_Dog_Driver

The LCD_Drive is a set of codes to implement a driver for the LCD dog module. The driver utilizes two arrays of 16 buffers to store the display data. Then the drive will write each byte of data to the LCD module using the SPI transmission. One must initialize the LCD module using the appropriate commands before they start writing the data. The process of initialization is pre-programmed, and the procedures can be found at the following flowchart and notes.

Notes:

*Internal OSC frequency adjustment is not necessary in this system.

* Function Set 0x39 for 2-line display; must be sent twice to the Controller.

* Bias Contrast is 0x1C for 2-line display.

* Power control is recommended to be 0x50.

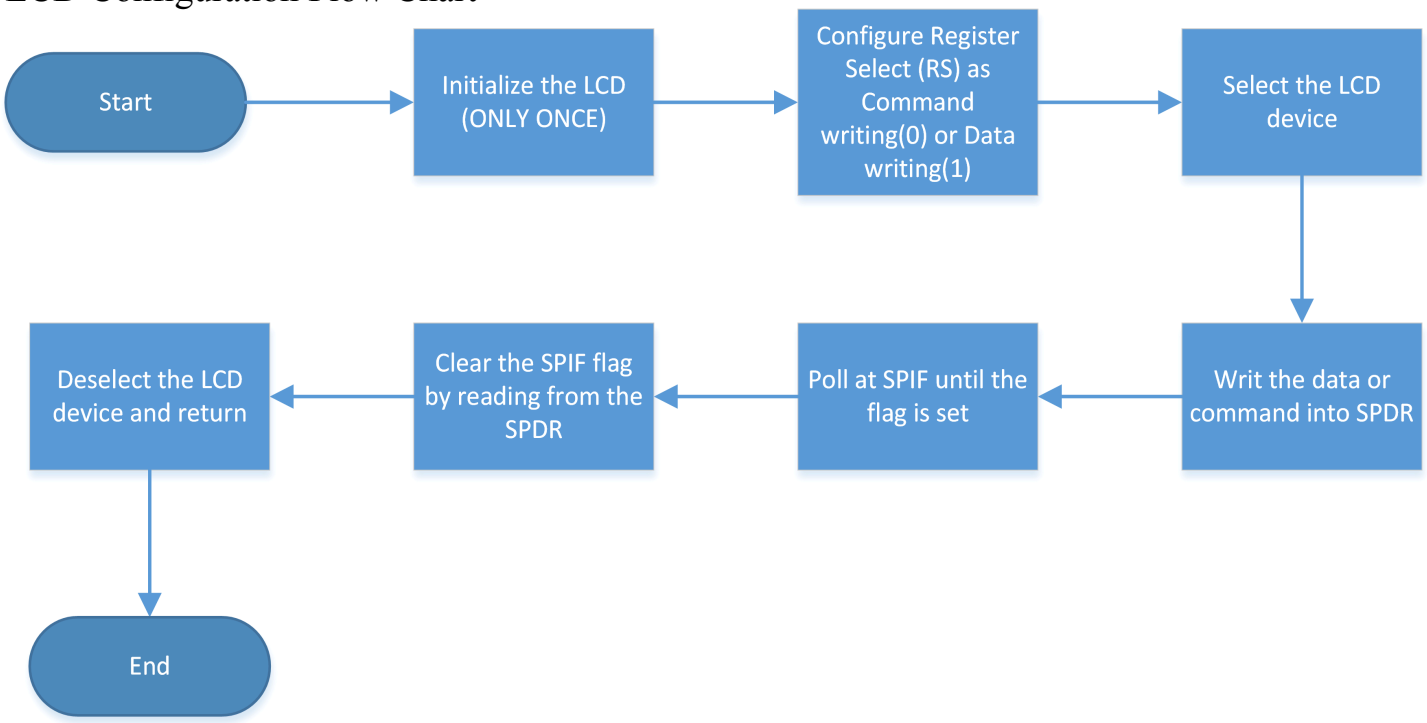
* Follower mode should be configured as on with 0x6C.

* Contrast is recommended to be 0x74, but adjustable according to user preference.

* This system will not show the cursor or the blink, the corresponding instruction is 0x0C for display ON/OFF control.

* Additional instructions to clear display or entry mode are 0x01 or 0x06, respectively.

LCD Configuration Flow Chart

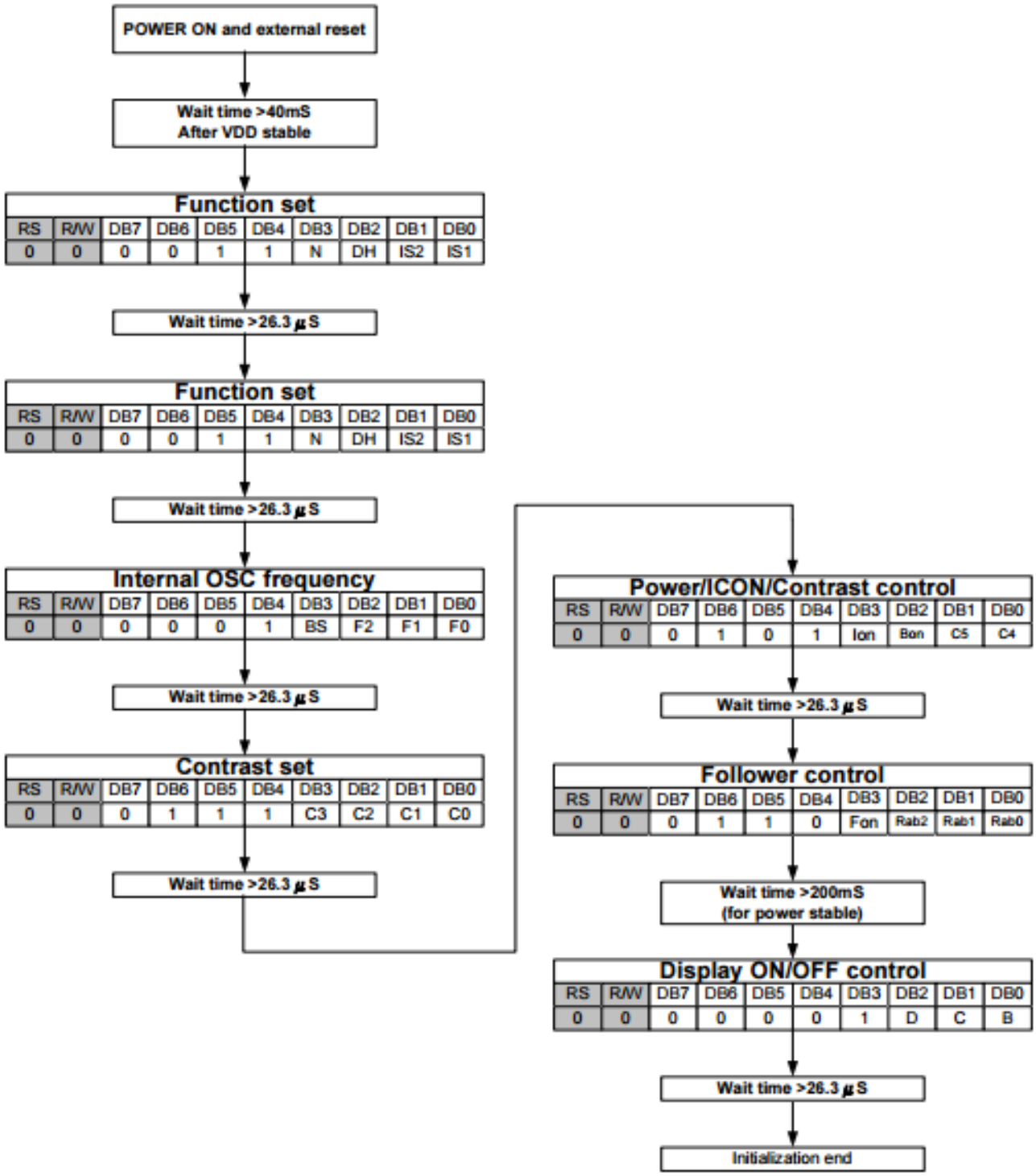


Flow Chart 5: program flow to configure the LCD display. *@ Copyright of VALI Inc.*

Additional Functions for LCD Operation/Configuration

Function	Description
void clear_dsp()	Use 3 for loops to empty all the display buffers, then reset the index to 0.
int putchar(int c)	This function override the putchar function in the c standard library. Since the system is required to display characters in the LCD screen instead of the Terminal window, the printf function will call the custom putchar function to load characters into two 16 characters arrays and update the contents in the arrays to the LCD screen. If both arrays are full, the program will simply roll back to the first index of the first array and overwrite the content in the buffer.

Serial Interface of ST7036 Dot Matrix LCD Controller/Driver



Flow Chart 6: This chart represents the configuration steps for the LCD display using the Dot Matrix LCD Controller/Driver . *Curtesy of ST7036 Dot Matrix LCD Controller/Driver Data Sheet.*

3.7 ADC161S626_driver

The ADC161S626 is Analog to Digit Converter with a 16-bit resolution. Since the ACD output is two's complement, the resultant ADC reading should have a precision of 15 bits. It ranges from 0x7FFF to 0x0000 for unipolar measurement used in this system. This system feature a ratio-metric approach to eliminate the noise effect in the Vcc. The assumption is given by:

$$V_1 * \frac{R_t}{R_t + R_{pullup}} = V_2 * \frac{ADC_{reading}}{2^{15}}$$

In the equation above, if V_1 and V_2 have the same voltage, they can be eliminated from the equation above. That yields a direct relation between the resistance ratio and the ADC readings.

Therefore, the voltage reference is served as the input to the temperature sensor circuitry in the actual design, while V_2 is using the same voltage reference from REF198.

Since the pull up resistor for the sensor circuitry is 10k, the maximum current driven from the REF 198 is given by:

$$I\_\max = \frac{V_{ref}}{R_{pullup}} = \frac{4.096V}{10kohm} = 0.4096\ mA$$

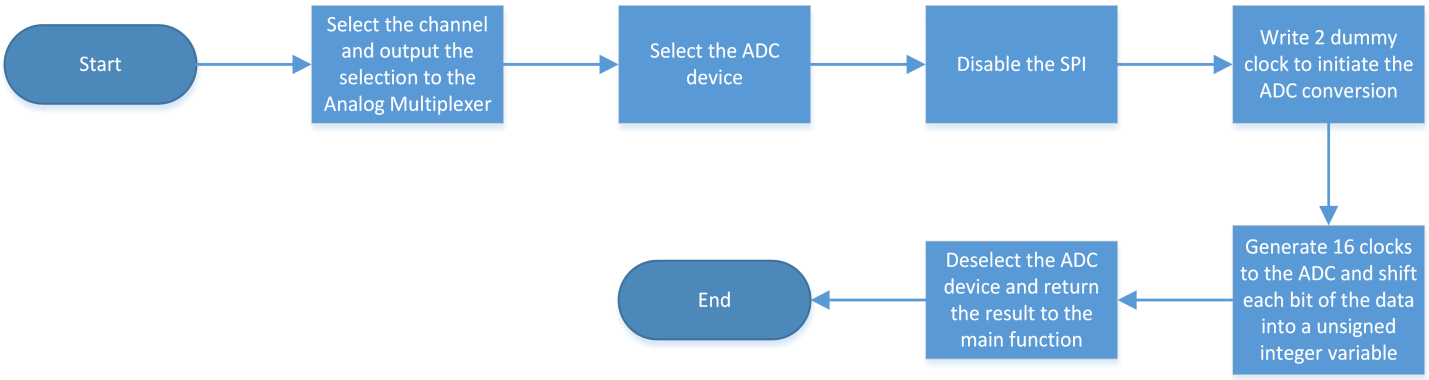
The resultant current is much less than the current limit of 30mA specified by the datasheet of the REF198, which proves that the V_ref is capable of serving as the input voltage.

Once the channel in the multiplexer is selected, the ADC will receive the analog voltage from the specific channel and return the ADC reading to the microcontroller unit (MCU) using a SPI (Serial Peripheral Interface). Since the hardware SPI is used by the LCD in this design, the system will use a software approach to implement this procedure. The steps are given by:

Select the ADC device (PORTA 0)
Disable the SPI hardware to avoid conflict.
Write two dummy clocks to initiate the ADC data transmission.
Write 16 clocks to the ADC SCK ports and shift all 16 bits data into an unsigned integer.
Deselect the ADC device and return the result to the main program.

The following flowchart gives a visual demonstration of the procedure.

Analog to digital Converter Flow Chart



Flow Chart 7: The program sends and selects the channel. Then the ADC conversion is initiated and store in the atmega128A. After the conversion and the data is stores, the ADC is deselected and the next function is initiated. *@ Copyright of VALI Inc.*

3.8 SPI Interface LCD driver and ADC driver

The software of this application relies on Serial Peripheral Interface (SPI) to transmit data. There are two approaches to implement the SPI in this system. The LCD driver uses the built-in SPI hardware from the microcontroller to send data to the LCD. The other part, the ADC driver, utilizes the software SPI to receive data from the ADC. Both approaches have important influences on the system.

SPI hardware (LCD driver)

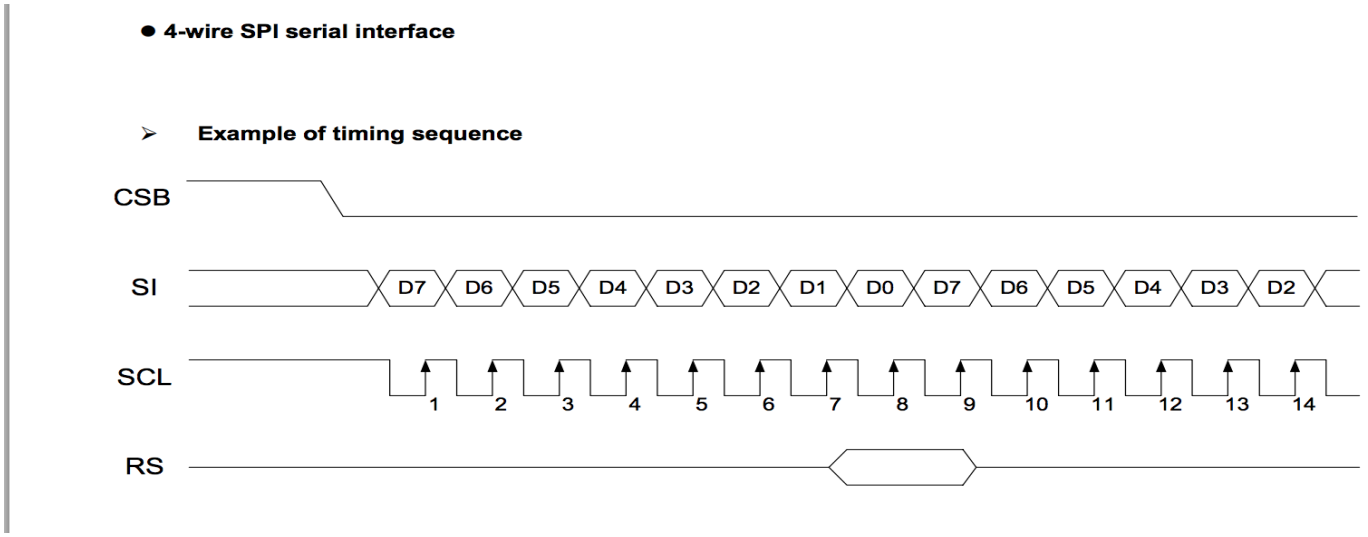


Diagram 1: Example timing sequence of SPI data transmitting. Curtesy of *ST7036 Dot Matrix LCD Controller/Driver Data Sheet.*

The diagram above shows the timing of the SPI data transmission. The commands or data to be sent to the LCD have the size of one byte and transmit time of 8 clocks. Therefore, the SPI hardware can be used to communicate with the LCD efficiently.

To configure the SPI hardware: the clock polarity is 1 since the clock is high when CSB is idle (SPOL = 1). The clock phase is also 1 because the data is sample at the trailing edge of the clock (SPOH = 1). The minimum clock period is 100ns from the datasheet, which indicates that the 16MHz clock of the microcontroller needed to be divided by a factor of 2 to provide the fastest system clock to the LCD controller. See the file LCD_driver.c for the full configuration of SPI hardware.

Curtesy of Atmega 182A & ST7036 Dot Matrix LCD Controller/Driver Data Sheet.

SPI hardware (LCD driver)

Timing Diagrams

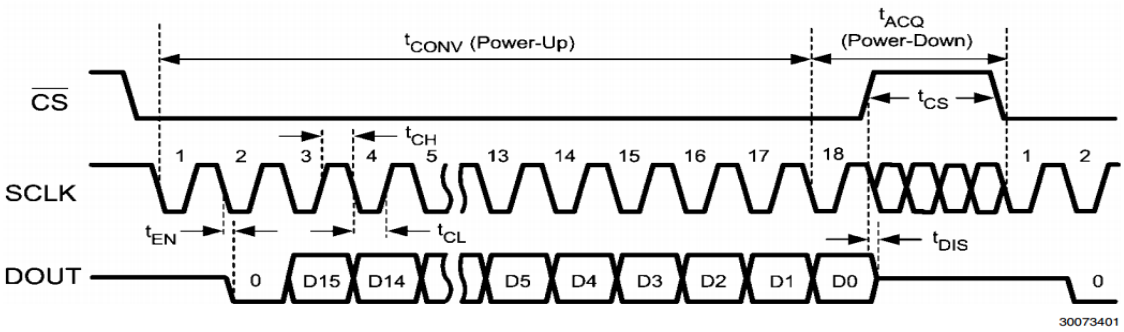


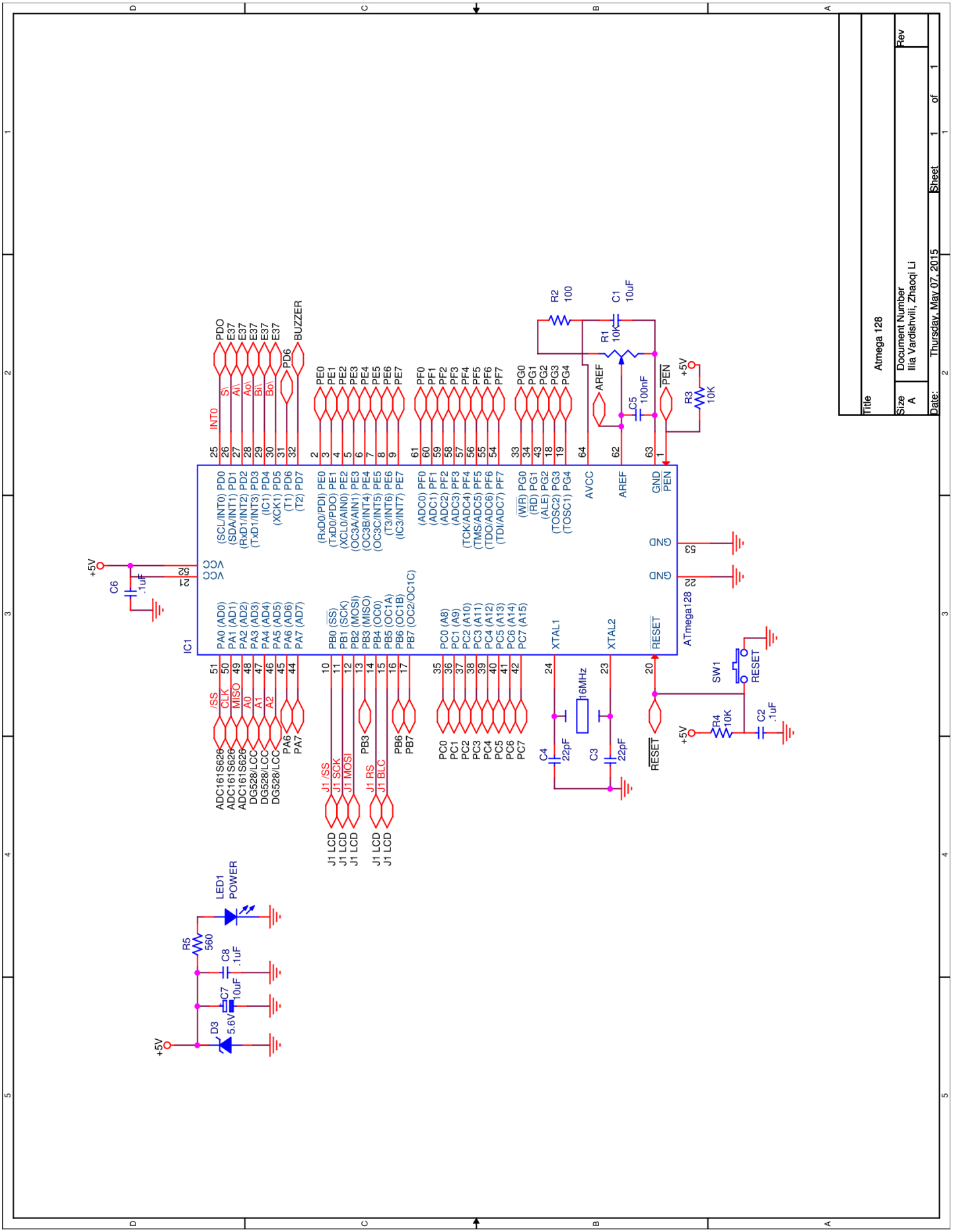
FIGURE 1. ADC161S626 Single Conversion Timing Diagram

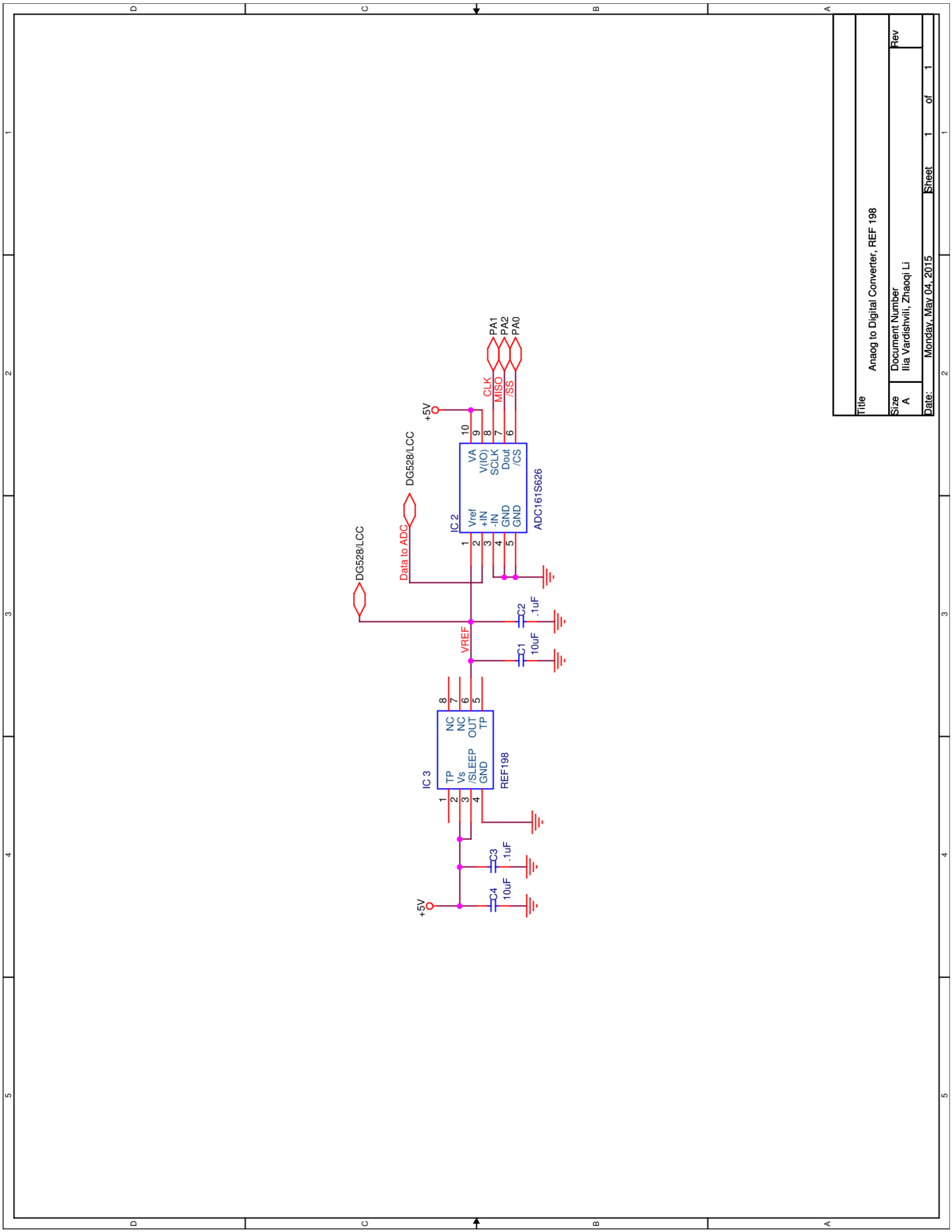
Diagram 2: Example timing sequence of SPI data transmitting. *Curtesy of ADC161626 16-bit Analog-to-Digital Converter datasheet*

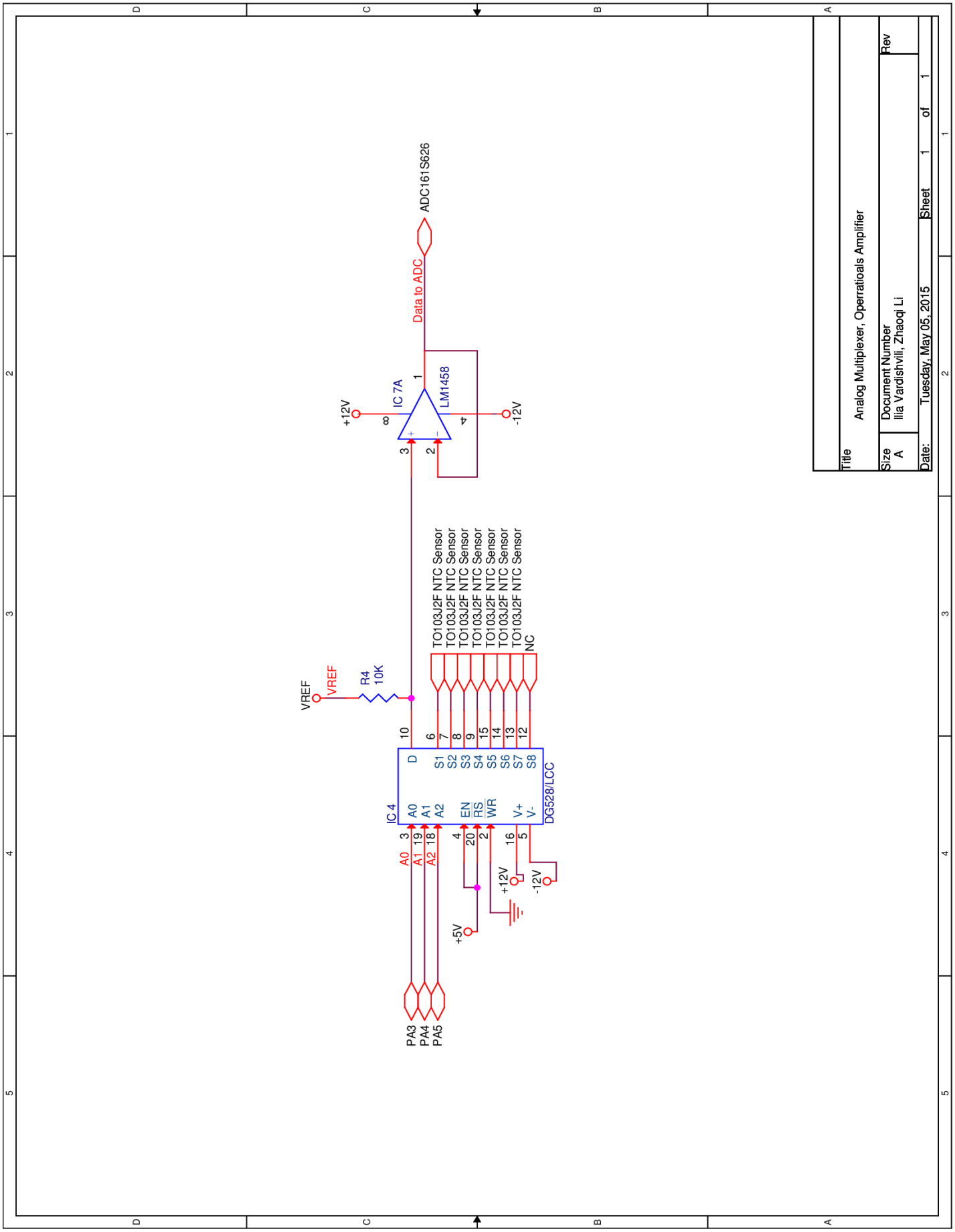
SPI software:

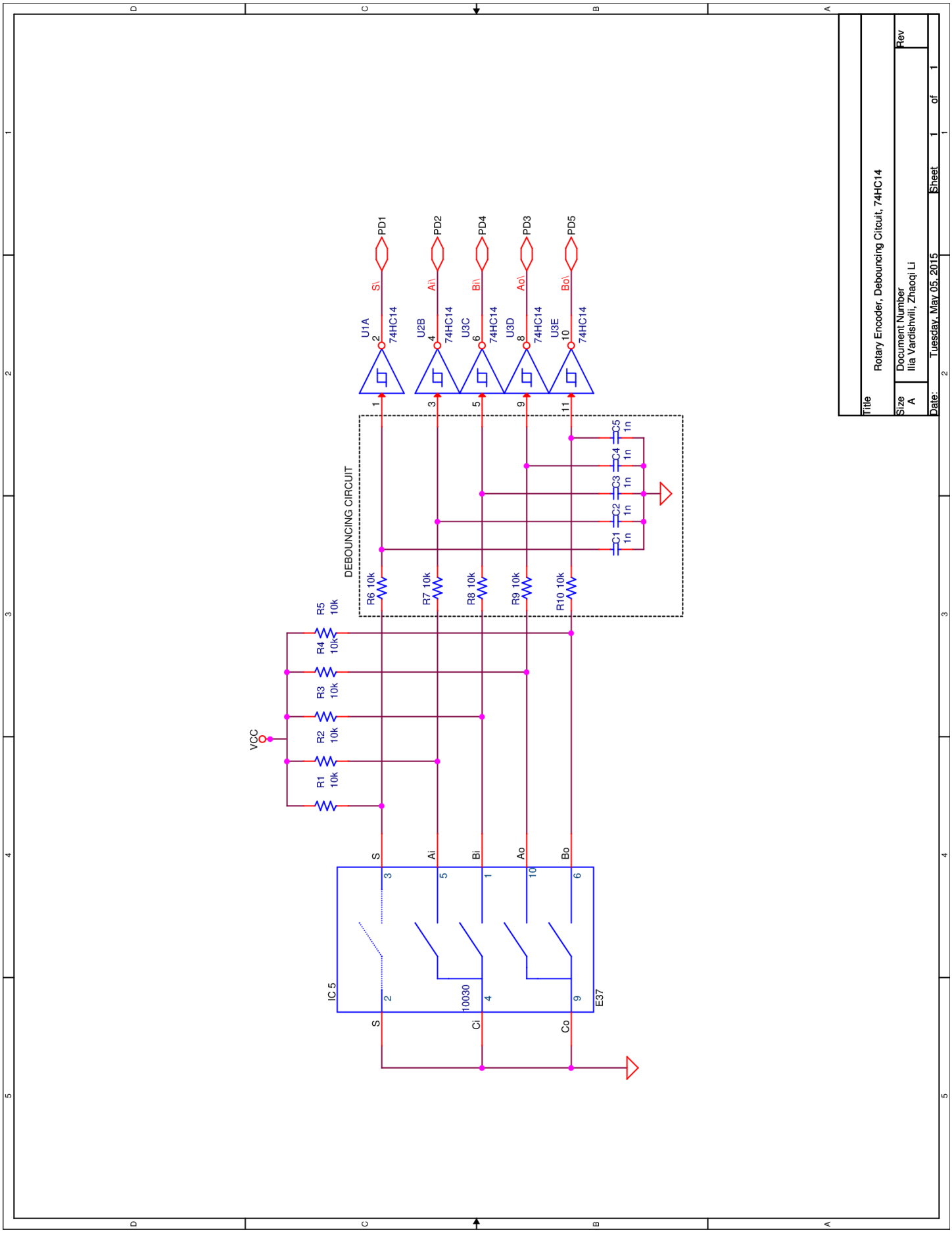
The software SPI approach is the so-called “bit-banging”, which uses the microcontroller to generate one clock with the proper period to transmit one bit of data. In the case of ADC161S626, the slave ADC needs to transmit 16 bits data to the microcontroller in 18 clock cycles, which violates the SPI hardware protocol that only generates 8 system clocks to receive one byte of data. Therefore, the “bit-banging” approach is used to transmit each bit of the data individually. The system will start with 2 dummy clocks to initiate the data transmission, then start receiving data at the third clock. See Flowchart 7 for a visual implementation and ADC161S626.c for the detailed implementation. *Curtesy of ADC161626 16-bit Analog-to-Digital Converter datasheet* .

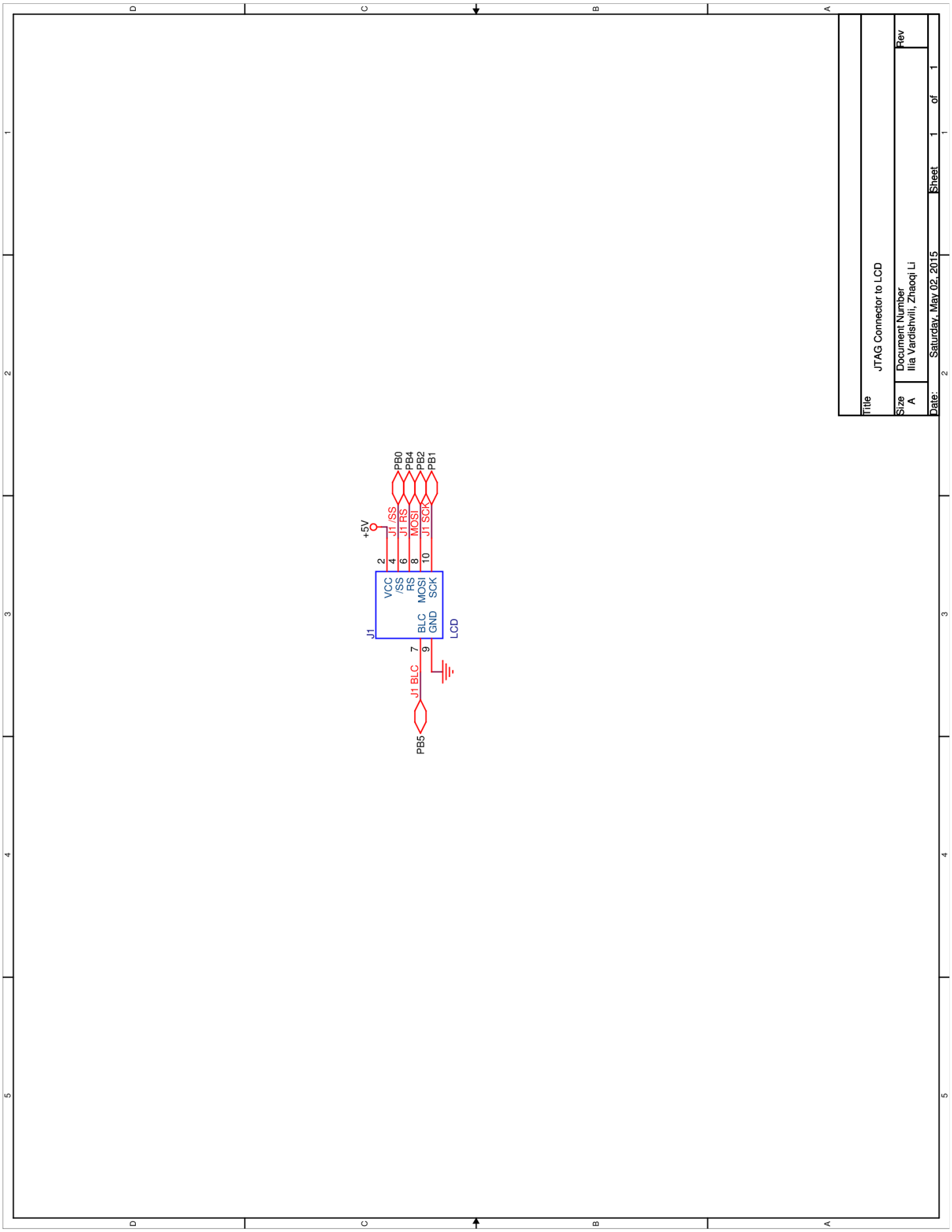
Chapter 4: Appendix A

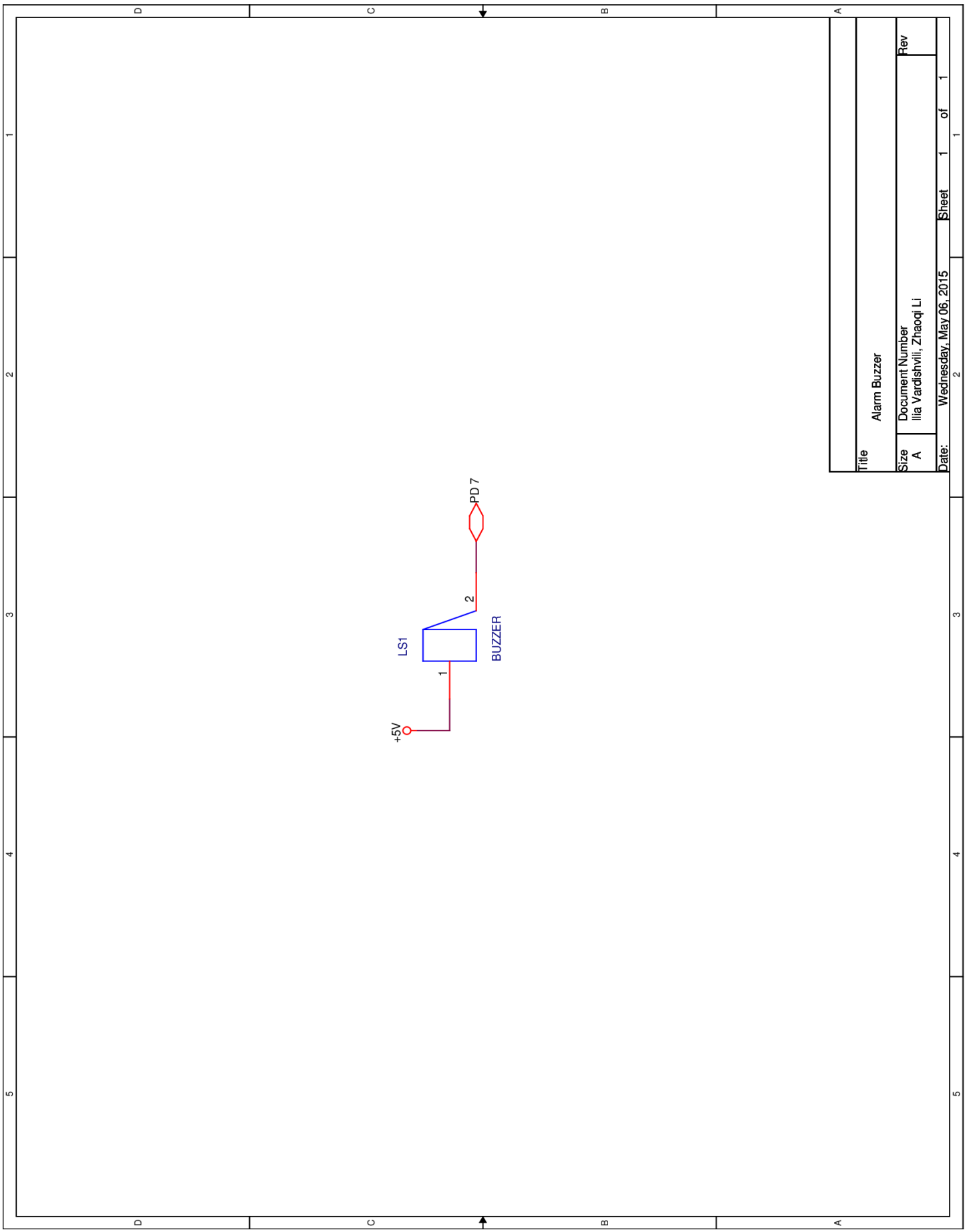












Chapter 5: Appendix B

```
/*****************************************************************************
 *
 * File Name             : main.c
 * Title                 : main file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION: This program includes a main function which is continously
 * performing background tasking. In the infinite function loop, in each second,
 * the program will take voltage readings from each channels 10 times and store
 * the volue into an array. Then the program will determine the state and select
 * the way to display the voltage result. In the auto mode, the temperature for each
 * channel will be refresh each second, but the channel value for reamin for 3
 * second and then display the temperature of the next channel for another 3 second.
 * In the man mode, the temperature will stay at 1 channel until a input cause the
 * program to display the readings from a different channel. Pressing the pushbutton
 * in the Man state will cause the system to enter the Limit state, in which the
 * user will be able to access and change the temperature limit for the current
 * channel. The temperature limit is to be stored in an array of character in the
 * eeprom. In the regulate Auot or Man state, if the measured temperature goes above
 * the limit, the system will generate a software interrupt in order to enter the
 * Alert state, in which the system will display an alert message and keep beeping the
 * buzzer. User can slient the buzzer for 8 second if the press the pushbutton to
 * escape into the Man state. If the temperature still exceed the limit, the system will
 * re-enter the Alert state and display the warning message. However, the buzzer will
 * not resume until the slience counter finishs counting.



 * This program will be using a table_driven fsm and interrupt inputs to regulates
 * all state changes.

 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 05: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) II
 *
 *****************************************************************************
 */
#include <iom128.h>
#include <avr_macros.h> //useful macros
#include <intrinsics.h> //useful intrinsic functions
#include <stdio.h>

#include "LCD_Driver.h"         //LCD
#include "ADC161S626_driver.h"  //ADC
#include "DG528_driver.h"       //MUX
```

```
#include "fsm.h"                //FSM

//function template
void limit_init();
extern signed int tb_lookup (unsigned int adc_data);
extern int putchar(int c);

//present state
extern state present_state;
extern char index;
// conversion result array
unsigned int channel_voltage[8][8];
//limit array
__eeprom signed char limit[8];
//temporary ADC and temperature storage
signed char temp_limit;
signed int temp_int;
static unsigned long int adc_sum;
//channel to conv and channel to display
unsigned char channel_conv = 0;
unsigned char channel_display = 0;      //default channel 0;
//temporary varaibles
static unsigned char i = 0, current_time = 0;
// variables to hold the value of voltage
signed int temperature = 0;
unsigned char temp_tenth = 0;
char unit_temp = 'C';
char flag = 0;
//loop counter to monitor clock
unsigned char counter;
//arrays to store strings
const char *a[] = {"Ambit.O", "Motor  ", "H.Contr", "D.Convt", "B.Box#1", "B.Box#2",
            "Ambit.I", "Chnl #7"};


void main(void){
  __disable_interrupt();
  //Anolog mux channel select output
  DDRA = 0x3B;

  DDRB = 0xFF;          //initialize LCD
  SETBIT(PORTB, 0);     //deselect slave
  init_lcd_dog();
  clear_dsp();

  DDRD = 0x01;                  //  initialize PortD for interrupt
  PORTD = 0x02;

  SETBIT(DDRD, 7);    //enable buzzer output
  SETBIT(PORTD, 7);   //disable buzzer

  EIMSK = 0x0F;                 // enable individual interrupt
  EICRA = 0x53;

  __enable_interrupt();
```

```
//infinite loop -- approximately 1 second for each loop
while(1){
  if(present_state == Limit){
    __EEGET(temp_limit,channel_display);
    if(flag & (1 << 0)){
      temp_int = ((signed int)(temp_limit)*9)/5 + 32;
      unit_temp = 'F';
    }else{
      temp_int = (signed int)temp_limit;
      unit_temp = 'C';
    }

    clear_dsp();
    printf("Limit: %d ",temp_int);
    putchar(unit_temp);
    update_lcd_dog();

  }else{
    for(channel_conv = 0; channel_conv < 8; channel_conv++){
      DG528_driver(channel_conv);        //select channel
      for(i = 0; i < 10; i++){  //conv for each channel 10 times
        channel_voltage[channel_conv][i%8] = ADC161_conv();
      }
    }

    if(present_state == Auto){
      if(current_time == 3){
        current_time = 0;      //reset timer;
        channel_display = (channel_display + 1)%8;
      }else
        current_time ++ ;
    }else
      current_time = 0;       //always reset timer on man state

    if(present_state == Alert && counter == 0)
      CLEARBIT(PORTD, 7);
    else{
      SETBIT(PORTD, 7);
      if(counter > 0)
        counter --;
    }
    //summing ADC reading for a particular channel
    adc_sum = 0;
    for(i = 0; i<8; i++)
      adc_sum += (unsigned long int)channel_voltage[channel_display][i];

    //average the sum and look up the temperature in the table
    temperature = tb_lookup(((unsigned int)(adc_sum >> 3)));

    if(flag & (1 << 0)){
      temperature = (temperature* 9)/5 + 320;
      unit_temp = 'F';
    }else
      unit_temp = 'C';
```

```
      if(temperature < 0)
         temp_tenth = (0 - temperature)%10;
      else
         temp_tenth = temperature%10;
      temperature = temperature/10;
      clear_dsp();
      printf("%s: %d.%d ", a[channel_display], temperature, temp_tenth);
      putchar(unit_temp);
      update_lcd_dog();
      __EEGET(temp_limit, channel_display);
       if(flag & (1 << 0))
          temp_int = ((signed int)(temp_limit)*9)/5 + 32;
       else
         temp_int = (signed int)temp_limit;

       if(((temp_int != 0) &&
          (temp_int <= temperature)) || present_state == Alert)
      {
        index = 16;
        printf("LIMIT EXCEEDED!!");
        update_lcd_dog();
        SETBIT(PORTD, 0);
        CLEARBIT(PORTD, 0);
      }
      __delay_cycles(8000000);    //delay_for_1 second
    }
  }
}
```

```
/*************************************************************************
 *
 * File Name             : fsm.h
 * Title                 : header file for the FSM
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION: This program includes the definition of typedef state, key,
 * function pointer, and a struct that forms the table.

 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 05: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) II
 *
 *************************************************************************
*/
// states in FSM, new states should be added after Man.
typedef enum {Auto, Man, Limit, Alert} state;

//All possible inputs to the FSM, new inputs may be added before the eol.
typedef enum {ocw, occw, icw, iccw, pbw, alarm, eol} key;

// pointers to function in a table
typedef void (* task_fn_ptr) ();

// self defined struct to construct a table
typedef struct{
  key keyval;
  state next_state;
  task_fn_ptr tf_ptr;
} transition;
```

```
/*****************************************************************************
 *
 * File Name             : FSM_ext.c
 * Title                 : main file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION: This program includes a fsm definition and a simple main function
 * which contains a sequence of input to change the behavior of the fsm. The fsm
 * has two major states(Auto and Man). Inputs of ocw and occw will cause the state
 * to toggle. This program will be using a table driven fsm approach.
 * Auto: occw and ocw will change the state to Man, other inputs do nothing.
 * All input will cause the fsm to execute the function null_fn() which simply
 * returns to the fsm.
 *
 * Man: occw and ocw will change the state to Auto, icw will trigger the function
 * incr_channel which increment the global varable channel. The value of channel
 * will roll over at 8. iccw will trigger the function decr_channel which
 * decrements the variable. The value will be reset to 7 at 0 decrement.
 *
 * Limit: pbw to Man state will cause the system to enter the limit state, in which
 * user can adjust or read the temeprature limit for the current channel. The temperature
 * limit will be saved into the eeprom. press the pushbutton again to exit.
 *
 * Alert: In the regulate Auot or Man state, if the measured temperature goes above
 * the limit, the system will generate a software interrupt in order to enter the
 * Alert state, in which the system will display an alert message and keep beeping the
 * buzzer. User can slient the buzzer for 8 second if the press the pushbutton to
 * escape into the Man state. If the temperature still exceed the limit, the system will
 * re-enter the Alert state and display the warning message. However, the buzzer will
 * not resume until the slience counter finishs counting.

 * call: incr_channel(), decr_channel(), null_fn(), start_counter, incr_limit_10,
 * incr_limit_1, decr_limit_10, decr_limit_1.
 *
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 04: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) I
 *
 *****************************************************************************
 */
#include "fsm.h"

//prototype of all FSM functions.
extern void incr_channel();
extern void decr_channel();
```

```
extern void incr_limit_10();
extern void decr_limit_10();
extern void incr_limit_1();
extern void decr_limit_1();
extern void start_counter();
extern void dsp_f();
extern void null_fn();

// Initial state after the reset
state present_state = Auto;



// The state transition of array consists of an array of arrays of structures.
// Each array of structure conrresponds to a particular present state value.
const transition auto_transitions[] =
{
// INPUT        NEXT_STATE       TASK
  {occw,        Man,             null_fn},
  {ocw,         Man,             null_fn},
  {alarm,       Alert,           null_fn},
  {pbw,         Auto,            dsp_f  },
  {eol,         Auto,            null_fn}
};

const transition man_transitions[] =
{
// INPUT        NEXT_STATe       TASK
  {occw,        Auto,            null_fn},
  {ocw,         Auto,            null_fn},
  {icw,         Man,             incr_channel},
  {iccw,        Man,             decr_channel},
  {pbw,         Limit,           null_fn},
  {alarm,       Alert,           null_fn},
  {eol,         Auto,            null_fn}
};

const transition limit_transitions[] =
{
// INPUT        NEXT_STATe       TASK
  {occw,        Limit,           decr_limit_10},
  {ocw,         Limit,           incr_limit_10},
  {icw,         Limit,           incr_limit_1},
  {iccw,        Limit,           decr_limit_1},
  {pbw,         Man,             null_fn},
  {eol,         Limit,           null_fn}
};

const transition alert_transitions[] =
{
// INPUT        NEXT_STATe       TASK
  {pbw,         Man,             start_counter},
  {eol,         Alert,           null_fn}
};
```

```
//outer array includes the pointer to the arrays of state transitions.
const transition *ps_transitions_ptr[4] =
{
  auto_transitions,
  man_transitions,
  limit_transitions,
  alert_transitions
};

// fsm function
void fsm(state ps, key keyval)
{

  int i;
  for(i=0; (ps_transitions_ptr[ps][i].keyval != keyval)
    && (ps_transitions_ptr[ps][i].keyval != eol); i++);

  ps_transitions_ptr[ps][i].tf_ptr();

  present_state = ps_transitions_ptr[ps][i].next_state;

}
```

```
/*****************************************************************************
 *
 * File Name             : fsm_function.c
 * Title                 : Header file for LCD module
 * Date                  : 02/07/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION
 * This file includes 3 fsm functions: incr_channel, decr_channel, null_fn
 * All functions take no parameter and return nothing. incr_channel will increment
 * the global valuable channel_display by 1 and roll over at 8. decr_channel will
 * decrement the channel_display by 1 and reset to 7 after 0. null_fn will simply
 * return and do nothing. incr_limit_10 and incr_limit_1 increases the limit,
 * decr_limit_10 and decr_limit_1 decreases the limit.
 *
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 4: table_driven fsm
 *
 *****************************************************************************
*/
#include <iom128.h>
#include <avr_macros.h> //useful macros
#include <intrinsics.h> //useful intrinsic functions

extern unsigned char counter;
extern char flag;
extern unsigned char channel_display;
extern __eeprom signed char limit[8];
signed char temp;

void incr_channel()
{
  channel_display = (channel_display + 1)%8;
}

void decr_channel()
{
  channel_display <= 0? channel_display = 7: --channel_display;
}

void null_fn()
{
  return;
}

void incr_limit_10(){
```

```
  __EEGET(temp, channel_display);
  if(temp + 10 > 94)
   __EEPUT(channel_display, 0);
  else
    __EEPUT(channel_display, temp+10);
}

void decr_limit_10(){
  __EEGET(temp, channel_display);
  if(temp < 10)
    __EEPUT(channel_display, 94);
  else
   __EEPUT(channel_display, temp-10);
}

void incr_limit_1(){
  __EEGET(temp, channel_display);
  if(temp == 94)
    __EEPUT(channel_display, 0);
  else
    __EEPUT(channel_display, temp+1);
}

void decr_limit_1(){
   __EEGET(temp, channel_display);
  if(temp == 0)
   __EEPUT(channel_display, 94);
  else
    __EEPUT(channel_display, temp-1); ;
}

void start_counter(){
  counter = 10;
  return;
}

void dsp_f(){
  flag ^= 1 << 0;
  return;
}
```

```
/*****************************************************************************
*
*  File Name            : LCD_Driver.h
*  Title                : Header file for LCD module
*  Date                 : 02/07/10
*  Version              : 1.0
*  Target MCU           : ATmega128 @  MHz
*  Target Hardware      ;
*  Author               : Ken Short
*  DESCRIPTION
*  This file includes all the declaration the compiler needs to
*  reference the functions and variables written in the files lcd_ext.c.
*  lcd.Driver.C
*
*  Warnings             : none
*  Restrictions         : none
*  Algorithms           : none
*  References           : none
*
*  Revision History     : Initial version
*
*
*****************************************************************************
*/

/**
 *  To use the lcd functions in lcd_dog_iar_driver.asm lcd_ext.c all that is
 *  needed is to include this file.
 */

/**
 *  This declaration tells the compiler to look for dsp_buff_x in
 *  another module. It is used by lcd_ext.c and main.c to locate the buffers.
 */
extern char dsp_buff_1[16];
extern char dsp_buff_2[16];
extern char dsp_buff_3[16];

/**
 *  Declaratios of low level lcd functions located in lcd_dog_iar_driver.asm
 *  Note that these are external.
 */
extern void init_lcd_dog(void);
extern void update_lcd_dog(void);

/**
 *  These functions are located in lcd_ext.c
 */
extern void clear_dsp(void);
extern int putchar(int);
```

```
/*****************************************************************************
 *
 * File Name             : LCD_Driver.c
 * Title                 : Header file for LCD module
 * Date                  : 02/07/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION
 * This file includes all the driver functions needed by the main function
 * to output messages to the lcd screen. The main functions will also need to
 * include <intrinsic.h> <avr_macro.h> in order to use all the functions in this
 * header. A multi-module version of this header function will be used next
 * time.
 *
 * public function: init_lcd_dog, update_lcd_dog, putchar, clear_dsp
 * local function: lcd_spi_transmit_CMD, lcd_spi_tranmit_DATA
 *
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 2: user interface
 *
 *****************************************************************************
 */
#include <iom128.h>
#include <intrinsics.h>     //Intrinsic functions.
#include <avr_macros.h>     //Useful macros.

/*
;
;*********************************************************************
;
;    *** Port B Interface Definitions:
;
;
;  Port B alt names    SCK    MISO   MOSI   /SS     RS      -      -      -
;  LCD Mod Signal      D6      -      D7    /CSB     -      -      -      -
;  LCD Mod Pin #       29      -      28     38      -      -      -      -
;
;       Notes:  RS ==>  0 = command regs, 1 = data regs
;              /SS = active low SPI select signal
;
;******************
*/
#define SCK     1
#define MISO    3
#define MOSI    2
#define SS_bar  0
```

```
#define RS      4
#define BLC     5

//Current Mode : 2 lines
   #define Addr_2 0xC0     //0x90 for 3 line display, 0xC0 for 2 lines
   #define Fun_set 0x39    // 0x39 for 2/3 lines, 0x35 for 1 line
   #define Bias_set 0x1C       //0x1C for 1/2 line, 0x1D for 3 lines

#define MAX_CHAR 16    // maximum character in a line

/*
 * Puts display buffers in near initialize to zero segment.
 * See chapter on segments in IAR reference guide.
 */
char dsp_buff_1[16];
char dsp_buff_2[16];
char dsp_buff_3[16];

//******* SPI transmission subroutines*********************************
/*
 *******************************
 NAME:       lcd_spi_transmit_CMD
 ASSUMES:    temp = byte for LCD.
             SPI port is configured.
 RETURNS:    nothing
 MODIFIES:   temp, PortB, SPCR
 CALLED BY:  init_dsp, update
 DESCRITION: outputs a byte passed in r16 via SPI port. Waits for data
             to be written by spi port before continuing.
 ***********************************************************************
*/

void lcd_spi_transmit_CMD(unsigned char temp){
   CLEARBIT(PORTB,RS);
   CLEARBIT(PORTB,SS_bar);
   SPDR = temp;
   while(!TESTBIT(SPSR,SPIF));
   SPSR &= 0x7F;
   SETBIT(PORTB, SS_bar);
 }

/*
;*******************************
 NAME:       lcd_spi_transmit_DATA
 ASSUMES:    temp = byte to transmit to LCD.
             SPI port is configured.
 RETURNS:    nothing
 MODIFIES:   temp, SPCR
 CALLED BY:  init_dsp, update
 DESCRITION: outputs a byte passed in r16 via SPI port. Waits for
             data to be written by spi port before continuing.
 ***********************************************************
 */
  void lcd_spi_transmit_DATA(unsigned char data){
    PORTB = (1 << RS);
```

```
    CLEARBIT(PORTB, SS_bar);
    CLEARBIT(SPSR, SPIF);
    SPDR = data;
    while(!TESTBIT(SPSR,SPIF));
    SPSR &= 0x7F;
    SETBIT(PORTB, SS_bar);
  }

/*
 ************************
 NAME:        init_lcd_dog
 ASSUMES:     nothing
 RETURNS:     nothing
 MODIFIES:    temp, SPSR, SPCR
 CALLED BY:   main application
 DESCRITION: inits DOG module LCD display for SPI (serial) operation.
 NOTE:  Can be used as is with MCU clock speeds of 4MHz or less.
 ***************************************************************
  public __version_1 void init_dsp(void)
*/


void init_lcd_dog(void){
    unsigned char temp;

    SPCR = 0x5F;                            // init SPI port for DOG LCD.
    SPSR &= ~(1 << SPIF);

    __delay_cycles(640000);                 //startup delay.

//func_set1
    temp = Fun_set;                           //send fuction set #1
    lcd_spi_transmit_CMD(temp);
    __delay_cycles(480);                    //delay for command to be processed

//func_set2
   temp = Fun_set;                          //send fuction set #2
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);                    //delay for command to be processed

//bias_set
   temp = Bias_set;                         //set bias value
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

//power_ctrl
   temp = 0x50;                           //0x50 nominal (delicate adjustment)
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

//follower_ctrl
   temp = 0x6C;                           //follower mode on...
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(640000);
```

```
//contrast_set
   temp = 0x74;                          //7C was too bright
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

//display_on
   temp = 0x0C;                          //display on, cursor off, blink off
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

//clr_display
   temp = 0x01;                          //clear display, cursor home
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

//entry_mode
   temp = 0x06;                          //clear display, cursor home
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

}

/*
 *************************
 NAME:       update_lcd_dog
 ASSUMES:    display buffers loaded with display data
 RETURNS:    nothing
 MODIFIES:   data,SREG, SPSR, SPCR

 DESCRITION: Updates the LCD display lines 1, 2, and 3, using the
   contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
 ***********************************************************************
 * public __version_1 void update_dsp_dog (void)
 */

void update_lcd_dog(void){
  unsigned char temp, i, data;

  SPCR = 0x5F;                            // init SPI port for DOG LCD.
  SPSR &= ~(1 << SPIF);

// snd_ddram_addr_1
  temp = 0x80;                           //init DDRAM addr-ctr
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

// snd snd_buff_1
  for(i = 0; i < MAX_CHAR; i++){
    data = dsp_buff_1[i];
    lcd_spi_transmit_DATA(data);
    __delay_cycles(480);
  }

// snd_ddram_addr_2
  temp = Addr_2;                          //init DDRAM addr-ctr
```

```
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

// snd snd_buff_2
  for(i = 0; i < MAX_CHAR; i++){
    data = dsp_buff_2[i];
    lcd_spi_transmit_DATA(data);
    __delay_cycles(480);
  }

// snd_ddram_addr_3
  temp = 0xD0;                          //init DDRAM addr-ctr
  lcd_spi_transmit_CMD(temp);
  __delay_cycles(480);

// snd snd_buff_3
  for(i = 0; i < MAX_CHAR; i++){
    data = dsp_buff_3[i];
    lcd_spi_transmit_DATA(data);
    __delay_cycles(480);
  }
}
//******* End Of SPI transmission *********************************//
```

```
//*****************************************************************************
//
// File Name            : lcd_ext.c
// Title                : LCD Utilities
// Date                 : 02/07/10
// Version              : 1.0
// Target MCU           : ATmega128 @  MHz
// Target Hardware      ;
// Author               : Ken Short
// DESCRIPTION
// The file contains two functions that make it easier for a C
// program to use the LCD display. The function clear_dsp() clears the display
// buffer arrays. When followed by the function update_dsp(), the
// display is blanked.
//
// The function putchar() puts a single character, passed to it as an argument,
// into the display buffer at the position corresponding to the value of
// variable index. This putchar function replaces the standard putchar funtion,
// so a printf statement will print to the LCD
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//
//*****************************************************************************

#include "LCD_driver.h"


char index;     // index into display buffer

//*****************************************************************************
// Function             : void clear_dsp(void)
// Date and version     : 02/07/10, version 1.0
// Target MCU           : ATmega128
// Author               : Ken Short
// DESCRIPTION
// Clears the display buffer. Treats each 16 character array separately.
// NOTE: update_dsp must be called after to see results
//
// Modified
//*****************************************************************************

void clear_dsp(void)
{
  // assuming buffers might not be contiguous
  for(char i = 0; i < 16; i++)
    dsp_buff_1[i] = ' ';

  for(char i = 0; i < 16; i++)
    dsp_buff_2[i] = ' ';
```

```
  for(char i = 0; i < 16; i++)
    dsp_buff_3[i] = ' ';

  index = 0;
}


//*****************************************************************************
// Function             : int putchar(int c)
// Date and version     : 02/07/10, version 1.0
// Target MCU           : ATmega128
// Author               : Ken Short
// DESCRIPTION
// This function displays a single ascii chararacter c on the lcd at the
// position specifieb by the global variable index
// NOTE: update_dsp must be called after to see results
//
//
// Modified
//*****************************************************************************

int putchar(int c)

{
  if (index < 16)
    dsp_buff_1[index++] = (char)c;

  else if (index < 32)
    dsp_buff_2[index++ - 16] = (char)c;

  else if (index < 48)
    dsp_buff_3[index++ - 32] = (char)c;

  else
  {index = 0;
  dsp_buff_1[index++] = (char)c;
  }

  return c;
}
```

```
/*****************************************************************************
 *
 * File Name             : DG528_driver.h
 * Title                 : main file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION: This header file includes a simple external function prototype
 * for the DG528 channel selection function. Include this file to use the
 * DG528_driver function.
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 04: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) I
 *
 *****************************************************************************
 */
//Mux channel select function

extern void DG528_driver(unsigned char channel);
```

```
/*************************************************************************
 *
 * File Name             : DG528_driver.c
 * Title                 : main file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION: This program defines a driver function to select the input channel
 * for the DG528 mux. The function is passed the valuable indicating the channel
 * selection. The function will then shift the value to the correct position in
 * order to allow the MCU to output a valid selection to the mux. Finally the
 * function will delay a period of time to allow the new voltage to settle down.
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 04: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) I
 *
 *************************************************************************
 */

#include <iom128.h>
#include <avr_macros.h>
#include <intrinsics.h>

void DG528_driver(unsigned char channel){
  channel = channel << 3;        //shift the value to the approriate position
  channel &= 0x38;              // mask unused bits

  PORTA &= 0xC7;
  PORTA |= channel;              //output selection to mux;
  __delay_cycles(1000);          //delay for voltage settle downo

  return;
}
```

```
/*****************************************************************************
 *
 * File Name             : ADC161S626_driver.h
 * Title                 : header file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION
 * This file contains the function prototype for the ADC161_conv.
 * The main program will include this file to use the ADC
 *
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 3: Single Channel Temperature Measurement Using NTC Sensor
 *
 *****************************************************************************
 */

//conversion function
extern unsigned int ADC161_conv();
```

```
/*****************************************************************************
 *
 * File Name             : ADC161S626_driver.c
 * Title                 : driver file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION
 * This file contains a driver for the ADC161S626. Once called, the function
 * will initialize the appropriate port for data transmission. Since this driver
 * will be using a software approach, it will disable the SPI function of the MCU
 * at the beginning. At the beginning of the transmission, the program will write
 * 2 dummy clock to allow the ADC to prepare the data. Then the program will write
 * 8 clocks to the device and read the MISO port to determin the data bit by bit.
 * At the end of the program, the MCU will deselect the device and return the result.
 *
 * inputs: MISO
 * outputs: SCK, SS_bar (all inputs and outputs are defiend through macros)
 *
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 3: Single Channel Temperature Measurement Using NTC Sensor
 *
 *****************************************************************************
 */

#include <iom128.h>
#include <avr_macros.h>

#define SS_bar PORTA_Bit0
#define SCLK PORTA_Bit1
#define MISO PINA_Bit2
#define SETUP DDRA
#define CONF  0x03


unsigned int ADC161_conv(){
  unsigned int result = 0;

  //SETUP = CONF;          //Configure PORT for data transmission


  SS_bar = 1;           // deselect device;
  SPCR = 0x00;          //disable SPI;
  SCLK = 1;             // clock polarity = 1, clock phase = 1

  SS_bar = 0;           //select device
```

```
  SCLK = 0;
  SCLK = 1;
  SCLK = 0;
  SCLK = 1;              // 2 dummy clocks to prepare data transmission

  for(char i=0; i < 16; i++){
    result = result << 1;
    SCLK = 0;            // load the next bit
    SCLK = 1;            // read the data at the leading edge

    if(MISO == 1)
      result |= 0x01;
  }

  SS_bar = 1;            //deselect device

  return result;
}
```

```
/*****************************************************************************
 *
 * File Name            : isr.c
 * Title                : main file for ADC161S626
 * Date                 : 03/24/10
 * Version              : 1.0
 * Target MCU           : ATmega128 16MHz
 * Target Hardware      ;
 * Author               : zhaoqi li, ilia vardishvili
 * DESCRIPTION: This file includes 2 ISR to receive inputs from the inner and
 * outer knob of the rotary encoder. INT2 will be responsible for the inner knob
 * and INT3 will be responsible for the outer knob. Clockwise turn will provide
 * cw input for the particular knob, ccw for counter clockwise turn. pbw for
 * pushbutton press. INIT0 is a software-generated interrupt which is triggered
 * by a positive pulse generated by PORTD 0
 * After reading the input, the ISR will call the fsm to determne the next state.
 * Warnings             : none
 * Restrictions         : none
 * Algorithms           : none
 * References           : none
 *
 * Revision History     : Initial version
 *
 * lab section 01, bench 04
 * Module 04: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) I
 *
 *****************************************************************************
 */
#include <iom128.h>
#include <avr_macros.h>
#include <intrinsics.h>
#include "fsm.h"

void check_release();
extern state present_state;
extern void fsm(state present_state, key key_input);
key key_input;

union{
  unsigned char byteImage;
  struct bitImage{
    unsigned char imbit0 : 1,
      imbit1 : 1,
      imbit2 : 1,
      imbit3 : 1,
      imbit4 : 1,
      imbit5 : 1,
      imbit6 : 1,
      imbit7 : 1;
  }temp;
};


/*
```

```
 * Interrupt serivce routine
 */
#pragma vector = INT0_vect
__interrupt void ISR_INT0(void)
{
  key_input = alarm;
  fsm(present_state, alarm);
}

#pragma vector = INT1_vect
__interrupt void ISR_INT1(void)
{
  key_input = pbw;
  fsm(present_state, key_input);
  check_release();
}

#pragma vector = INT2_vect
__interrupt void ISR_INT2(void)
{
  byteImage = PIND;
  if(temp.imbit4^temp.imbit2)
    key_input = icw;
  else
    key_input = iccw;
  fsm(present_state, key_input);
}

#pragma vector = INT3_vect
__interrupt void ISR_INT3(void)
{
  byteImage = PIND;
  if(temp.imbit5^temp.imbit3)
    key_input = ocw;
  else
    key_input = occw;
  fsm(present_state, key_input);
}

/*
*  Check keypad is released and not bouncing.
*/
void check_release(void)
{
  while(!TESTBIT(PIND,INT1));     //Check that keypad key is released.

  __delay_cycles(50000);          //Delay (.05secs) / (1 / 1MHz) cycles.

  while(!TESTBIT(PIND,INT1));     //Check that key has stopped bouncing.
  return;
}
```

```
/*****************************************************************************
 *
 * File Name             : fcn_tb.c
 * Title                 : main file for ADC161S626
 * Date                  : 03/24/10
 * Version               : 1.0
 * Target MCU            : ATmega128 16MHz
 * Target Hardware       ;
 * Author                : zhaoqi li, ilia vardishvili
 * DESCRIPTION: a function that provides a table look for the input ADC reading
 * then return the corresponding temperature to the caller.
 * DG528_driver function.
 * Warnings              : none
 * Restrictions          : none
 * Algorithms            : none
 * References            : none
 *
 * Revision History      : Initial version
 *
 * lab section 01, bench 04
 * Module 05: Table Driven Finite State Machine (FSM) Control of an Electric
 * Car Temperature Monitoring System (ECTMS) II
 *
 *****************************************************************************
 */
#include <stdio.h>
#define FULL_SCALE 32768
const unsigned long int table_1[] =
{
176974, 166356, 156441, 147177, 138518, 130421, 122847, 115759,
109122, 102906, 97081, 91621, 86501, 81698, 77190, 72957, 68982,
65246, 61736, 58434, 55329, 52407, 49656, 47066, 44626, 42327,
40159, 38115, 36187, 34368, 32650
};
const unsigned int table_2[] =
{
31029, 29498, 28052, 26685,
25392, 24170, 23013, 21918,
20882, 19901, 18971, 18090,
17255, 16463, 15712, 14999,
14323, 13681, 13072, 12493,
11942, 11419, 10922, 10450,//54
10000, 9572,  9165,  8777,
8408,  8057,  7722,
};
const unsigned int table_3[] =
{
7402, 7098, 6808, 6531, 6267, 6015, 5775, 5545,
5326, 5117, 4917, 4725,4543, 4368, 4201, 4041,
3888, 3742, 3602, 3468, 3340, 3217, 3099, 2986,
2878, 2774, 2675, 2579, 2488, 2400, 2316
};
const unsigned int table_4[] =
{
2235, 2157, 2083, 2011, 1942, 1876, 1813, 1752,
```

```
1693, 1637, 1582, 1530, 1480, 1432, 1385, 1340,
1297,1255, 1215, 1177, 1140, 1104, 1070, 1037,
1005, 973,  944,  915,  887,  861,  835,  810
};

signed int tb_lookup (unsigned int adc_data)
{
  char i;
  signed int index = 125;
  unsigned long int var_a, var_b;
  unsigned long int result;
  result = (unsigned long int)(adc_data)* 10000/(FULL_SCALE - adc_data);
  //check first array
  if(table_1[30] < result){
    for(i = 0; i < 31; i++){
      if(result > table_1[i]){
        index = i;
        if(i != 0){
          var_a = table_1[i-1];
          var_b = table_1[i];
        }
        break;
      }
    }
    }
  else if(table_2[30] < result){
     for(i = 0; i < 31; i++){
      if(result > table_2[i]){
        index = 31 + i;
        if(i == 0){
          var_a = table_1[30];
          var_b = table_2[0];
        }else{
          var_a = table_2[i-1];
          var_b = table_2[i];
        }
        break;
      }
     }
      }
  else if(table_3[30] < result){
     for(i = 0; i < 31; i++){
      if(result > table_3[i]){
        index = 62 + i;
        if(i == 0){
          var_a = table_2[30];
          var_b = table_3[0];
        }else{
          var_a = table_3[i-1];
          var_b = table_3[i];
        }
        break;
      }
     }
      }
```

```
  else{
     for(i = 0; i < 30; i++){
      if(result > table_4[i]){
        index = 93 + i;
         if(i == 0){
          var_a = table_3[30];
          var_b = table_4[0];
        }else{
          var_a = table_4[i-1];
          var_b = table_4[i];
        }
        break;
      }
     }
  }


 if(index == 125)
   return 940;
  else if (index == 0)
    return -300;
  else{
    index = (index - 30)*10;
    index -= ((result - var_b)*10)/(var_a - var_b);
    return index;
  }

}
```

End of VALI-VI381 Thermal Monitoring System Design Document