

EGGBOT

DOMINIC DOTY

The Pursuit of the Perfect Egg

Professor Linden McClure Ph.D.
Embedded System Design
University of Colorado, Boulder
May 5 2019

HONOR CODE PLEDGE

By signing this sheet, students certify that the work they submit is their own, and that they have clearly identified any code, schematics, design details, documentation, pictures, or other work obtained from another source. Each student on a project team is expected to certify their work by signing this pledge.

HONOR CODE PLEDGE: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

Dominic Doty
Boulder Colorado, May 5 2019

ABSTRACT

A final project for the Embedded System Design course at the University of Colorado, Boulder. The Eggbot is a single axis robot with a temperature controlled water bath created for the goal of cooking the perfect soft boiled egg. The bot also serves a secondary purpose of precision tea brewing, and a tertiary purpose of heating water to specific temperatures.

The stated course goal of the project was to explore an area of interest while implementing new hardware and software. My personal goal extends beyond this, to create a consumer quality product that can be used every day in my home, while also showcasing the development board created through the duration of this course.

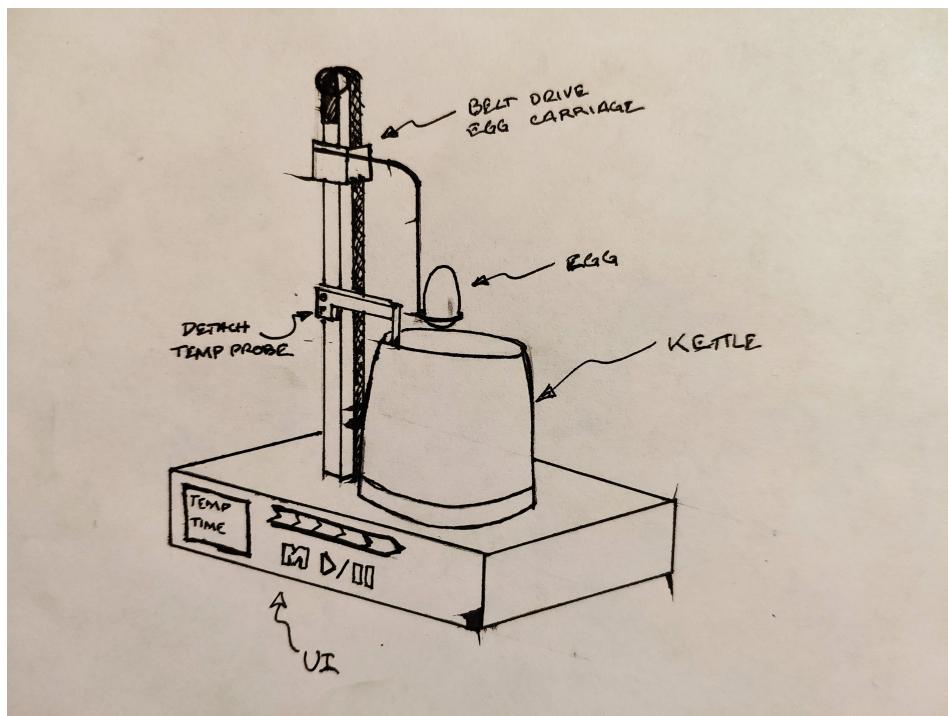


Figure 0.1: Concept Sketch of the EggBot.

CONTENTS

1 DESIGN	1
1.1 Hardware Design	1
1.1.1 System Overview	1
1.1.2 Relay	2
1.1.3 Temperature Sensing	3
1.1.4 Stepper Driver	5
1.1.5 End-stop	5
1.1.6 Capacitive Touch	6
1.2 Firmware Design	7
1.2.1 System Overview	7
1.2.2 Scheduling Timer	8
1.2.3 User Interface State Machine	9
1.2.4 Display Task & LCD Driver	10
1.2.5 Step Generator & Output Routine	10
1.2.6 Button Task & Capacitive Touch Driver	11
1.2.7 Thermal Control & ADC Driver	12
1.2.8 Timer Task	12
1.2.9 Endstop Interrupt Service Routine	12
1.3 Mechanical Design	13
1.3.1 Design Overview	13
2 IMPLEMENTATION & VERIFICATION	15
2.1 Hardware Implementation & Verification	15
2.1.1 System Overview	15
2.1.2 Relay	15
2.1.3 Temperature Sensing	16
2.1.4 Stepper Driver	16
2.1.5 Endstop	16
2.1.6 Capacitive Touch	17
2.2 Firmware Implementation & Verification	20
2.2.1 System Overview	20
2.2.2 Scheduling Timer	20
2.2.3 User Interface State Machine	22
2.2.4 Display Task & LCD Driver	23
2.2.5 Step Generator & Output Routine	23
2.2.6 Button Task & Capacitive Touch Driver	25
2.2.7 Thermal Control & ADC Driver	25
2.2.8 Timer Task	26
2.2.9 Endstop Interrupt Service Routine	27
2.3 System Level Verification	27
3 CONCLUSION	28
3.1 Lessons Learned	28
3.2 Future Developments	29

3.3	Conclusion	29
4	ACKNOWLEDGMENTS	30
A	BILL OF MATERIALS	31
B	SCHEMATICS	32
C	CODE	41
D	PART DATASHEETS	91
D.1	Relay Datasheet	91
D.2	ADC Datasheet	96
D.3	Capacitive Touch IC Datasheet	131
D.4	Endstop Datasheet	182
D.5	Operational Amplifier Datasheet	187

LIST OF FIGURES

Figure 0.1	Concept Sketch of the EggBot	iii
Figure 1.1	System Hardware Block Diagram	1
Figure 1.2	Relay Driver Circuit	2
Figure 1.3	Relay Connection	3
Figure 1.4	Thermistor Linearity	4
Figure 1.5	ADC and Filter Schematic	4
Figure 1.6	End-stop Debouncing Schematic	5
Figure 1.7	Capacitive Touch Breakout Schematic	7
Figure 1.8	Capacitive Touch Breakout PCB	7
Figure 1.9	System Software Block Diagram	8
Figure 1.10	User Interface State Machine	9
Figure 1.11	CAD Render of Eggbot	13
Figure 2.1	Debouncing Circuit Logic Analyzer Capture	17
Figure 2.2	Capacitive Touch V1 Board	17
Figure 2.3	Capacitive Touch V2 Board	18
Figure 2.4	Populated Capacitive Touch Board	19
Figure 2.5	Capacitive Touch Electrode Construction	19
Figure 2.6	Stepper Pulse Generation Logic Capture	24
Figure 2.7	Stepper Movement Plot	24
Figure 2.8	Modified Bang-Bang Control Plot	26

LIST OF TABLES

Table A.1	Bill of Materials	31
-----------	-----------------------------	----

LISTINGS

Listing 2.1	Snippet of Scheduler Interrupt Service Routine (ISR)	20
Listing 2.2	Snippet of Main Scheduler	21
Listing 2.3	User interface pseudo code.	22
Listing 2.4	Snippet of Capacitive Touch Driver.	25
Listing C.1	Pin Settings Header	42
Listing C.2	Main Source	43
Listing C.3	Scheduler Header	60

Listing C.4	Scheduler Source	61
Listing C.5	Stepper Motor Driver Header	63
Listing C.6	Stepper Motor Driver Source	64
Listing C.7	Enstop Header	76
Listing C.8	Endstop Driver Source	76
Listing C.9	SPI Driver Header	78
Listing C.10	SPI Driver Source	79
Listing C.11	Capacitive Touch Driver Header	81
Listing C.12	Capacitive Touch Driver Source	82
Listing C.13	ADC Driver Header	84
Listing C.14	ADC Driver Source	84
Listing C.15	Thermistor Driver Header	87
Listing C.16	Thermistor Driver Source	87
Listing C.17	Debug Printing Header	89
Listing C.18	Delay Functions Header	90
Listing C.19	Delay Functions Source	90

ACRONYMS

ADC	Analog to Digital Converter
AT42	Capacitive Touch Sensor AT42QT110
AT89	80C52 Microcontroller AT89C51RC2
BCD	Binary Coded Decimal
COTS	commercial off the shelf
DIP	Dual Inline Package
DRV	Stepper Driver DRV8825
ESD	Embedded System Design
ISR	Interrupt Service Routine
LCD	Liquid Crystal Display
mA	milliamperes
MCAD	Mechanical Computer Aided Design
PCB	Printed Circuit Board
QFN	Quad Flat No-lead
SPI	Serial Peripheral Interface

UART Universal Asynchronous Receiver and Transmitter

VAC Volts Alternating Current

DESIGN

This chapter covers the design of the Eggbot with specific information about each subsystem of the bot including electrical, firmware, and mechanical design. Design trade-offs, component selection, and design intent are covered. As previously mentioned, my principal goal in the creation of the Eggbot was to achieve a polished, consumer quality result that could showcase the board I spent the semester creating.

1.1 HARDWARE DESIGN

1.1.1 System Overview

The electrical aspects of the project can be classified into three main categories: thermal control, motion control, and user interface. Thermal control consists of a commercial off the shelf ([COTS](#)) kettle, connected to 120 Volts Alternating Current ([VAC](#)) by a relay. Motion control is made up of a stepper motor and a Stepper Driver DRV8825 ([DRV](#)) based driver board. User interface includes a capacitive touch keypad, based on the Capacitive Touch Sensor AT42QT110 ([AT42](#)), and a Liquid Crystal Display ([LCD](#)), implemented in previous labs. The center of the electrical system is the 80C52 Microcontroller AT89C51RC2 ([AT89](#)) microcontroller board. The microcontroller will not be covered in detail in this report, since it is the standard board built in the Embedded System Design ([ESD](#)) course. The system block diagram is shown in Figure 1.1.

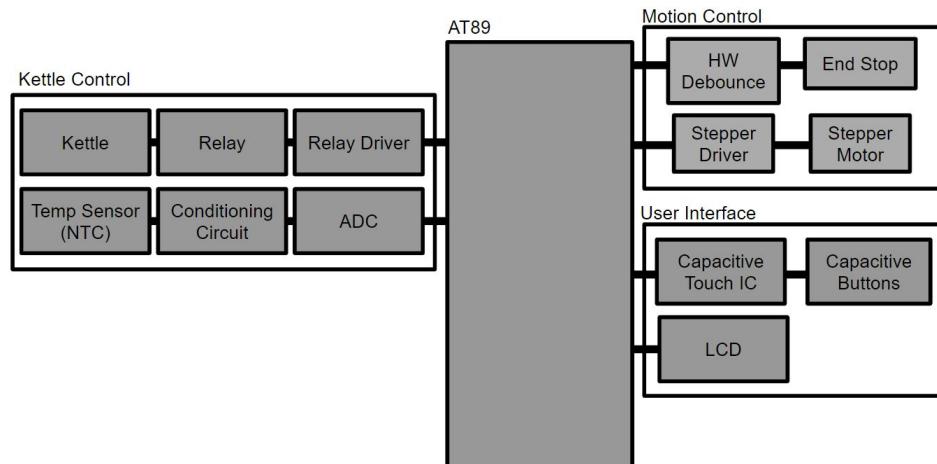


Figure 1.1: System Hardware Block Diagram.

1.1.2 Relay

The relay selected is an RZF1-1A4-L005 from TE Connectivity (D.1). This relay was selected because it features blade terminals for the VAC connected side of the relay, and through hole pins for the coil side of the relay. This eases connection of the relay to thick gauge, high voltage wire. It also provides some physical isolation between the low voltage control pins and the high voltage terminals. This increases system safety since high voltages need not be broken out on the same Printed Circuit Board (PCB) as low voltage signals.

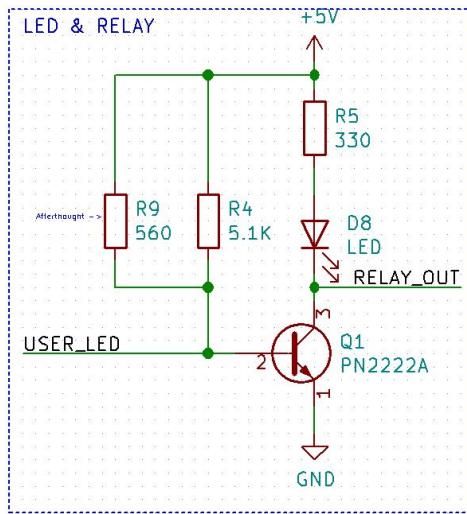


Figure 1.2: Relay Driver Circuit.

The relay is driven by a transistor from the microcontroller as shown in Figure 1.2. The relay was added here to utilize the partially implemented circuitry, and to take advantage of the LED as an indicator of power being supplied to the relay. The transistor base is driven by two pull-up resistors because the original pull-up resistor was sized to light the LED included in the circuit. With the addition of the relay, the pull-up resistor needed to be resized to supply adequate current to the transistor, and it was more efficient to add a parallel resistor than to remove the original one. The pull-up is sized to supply 10 milliamperes (mA) to the base of the transistor, enough to fully turn on the transistor with the LED current of 6mA and a coil current of 12mA.

This is admittedly somewhat aggressive since the maximum sink current for one microcontroller pin is 10mA, but I deemed it necessary to drive the transistor as hard as possible since the coil is an inductive load and will present much higher currents at turn on. To eliminate the inductive kick from the relay coil, a 1N4007-TPMSCT-ND diode was connected across the relay coil as a snubber diode as shown in Figure 1.3.

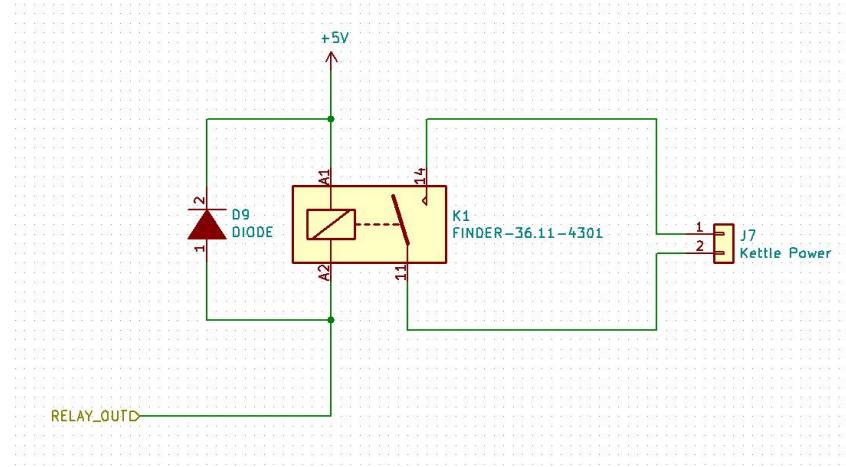


Figure 1.3: Relay connection itself.

1.1.3 Temperature Sensing

Temperature sensing is provided by a 10K thermistor (unknown brand), analog filtering circuitry, and a MCP3002-I/P Analog to Digital Converter ([ADC](#)) from Microchip Technology ([D.2](#)).

The thermistor is linearized by using it as one leg of a resistor divider (figure [1.5](#)). The linearizing resistor was selected using Figure [1.4](#) as a reference. In this figure, it can be seen that a resistor value of $2.7\text{ k}\Omega^1$ would yield a mostly linear reading from 20°C to 90°C . Since this is the region of interest for cooking with water just under boiling, this linear range works well².

After linearization, the sense voltage is buffered with a MCP602-I/P operational amplifier from Microchip Technology ([D.5](#)). This allows the sense voltage to be sampled with a high impedance input, and driven to a low impedance output. This is essential for driving the low-pass filter and [ADC](#) that follow. This amplifier in particular was selected because it is a rail-to-rail amplifier, capable of driving its output voltage very near to both ground and V+ supply.

The low-pass filter was sized starting with the capacitor on hand, a $47\mu\text{F}$ electrolytic. A cutoff frequency target was set at 40Hz to attempt to attenuate AC noise from the kettle at 60Hz. With this target, a resistance of 82Ω was selected, yielding a cutoff frequency of 41.3Hz.

While the capacitor value chosen for the filter may seem like overkill, it also helps with the sampling of the [ADC](#). The [ADC](#) uses a sample and

¹ R_{25} is the resistance at 25°C , in this case 10 k Ω

² Recall that the boiling point in Boulder, Colorado is approximately 95°C

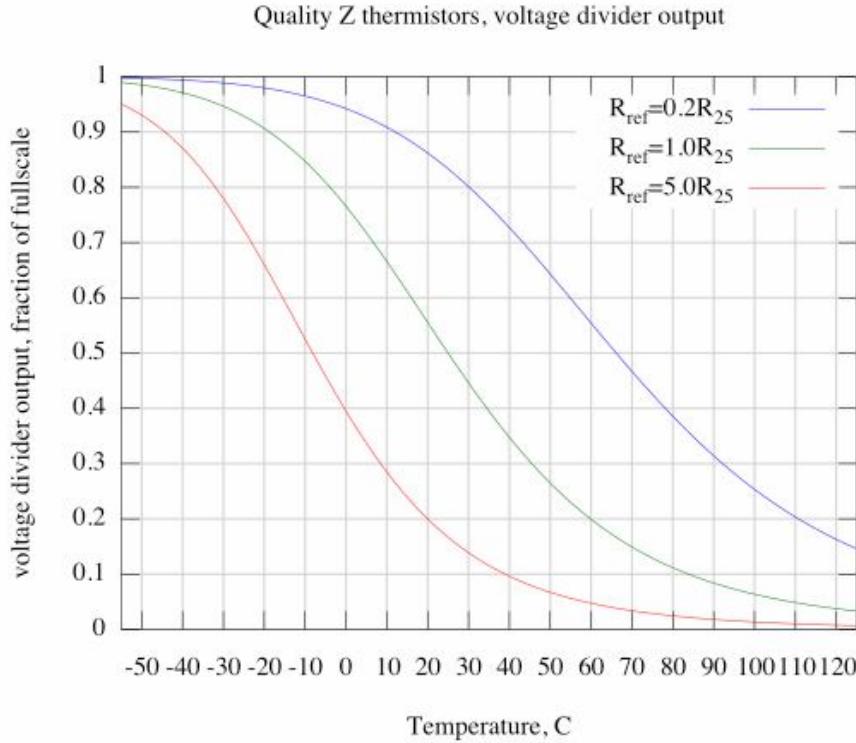


Figure 1.4: Thermistor linearity in a resistor divider for multiple resistor divider values. *Thermistor signal conditioning: Dos and Don'ts, Tips and Tricks –* <https://www.embeddedrelated.com/showarticle/91.php>

hold front end which appears as a small capacitor being rapidly connected and disconnected from the line. The sampled voltage must have a low impedance source so the **ADC** sampling does not significantly affect the sampled voltage. The combination of the op amp buffer and large capacitance effectively drive the **ADC** input.

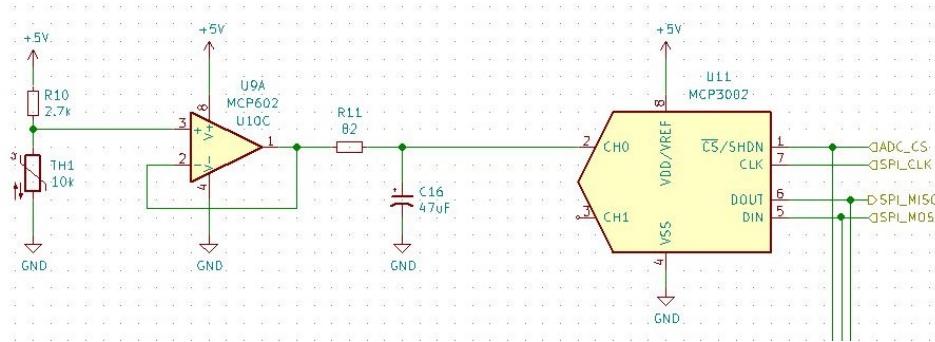


Figure 1.5: Thermistor linearization, filtering, and **ADC** schematic.

The specific **ADC** was selected for its Serial Peripheral Interface (**SPI**), low cost, Dual Inline Package (**DIP**) form factor, and 10-bit resolution.

The SPI interface was preferable because the AT89 includes a hardware SPI peripheral. While the AT89 also has a hardware Universal Asynchronous Receiver and Transmitter (UART), this is used for system programming and debugging.

1.1.4 Stepper Driver

The stepper motor and its driver were components that I had on hand at the beginning of the project, so their selection was not subjected to much scrutiny. The DRV8825³ is a common chip available cheaply on small breakout boards. The breakout board includes a current limiting potentiometer and some decoupling capacitors.

1.1.5 End-stop

For the end-stop, a microswitch D2FS-FL-N from Omron was selected (D.4). The end-stop was selected for its low cost and high durability. It is interesting to note standards of durability and usage are very different depending on the engineer's perspective. This switch is billed as "Developed for a few number of operations [sic] during a long period of use"⁴, which for an industrial context translates to a minimum lifetime of 10,000 operations. Needless to say, 10,000 operations far exceeds the expected lifetime of my system.

Since the end-stop is being connected directly to an interrupt pin on the processor, it requires further processing before connection. Enter the debouncing and triggering schematic shown in Figure 1.6.

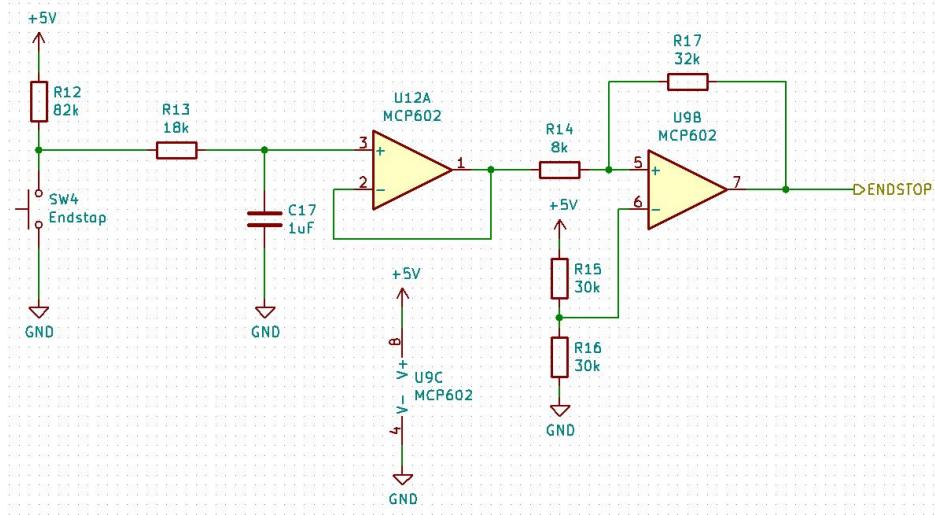


Figure 1.6: End-stop debouncing and triggering circuit.

³ <https://www.pololu.com/product/2133>

⁴ D.4

The left side of the schematic forms the debouncing circuit, with R₁₂ acting as a pull-up, and R₁₃ and C₁₇ forming a low-pass filter⁵. U₁₂ is another MCP602 operational amplifier (D.5) acting as a buffer to provide a low impedance source to the next part of the circuit.

The right side of the schematic forms a Schmitt Trigger. With the chosen values of resistors⁶, the trigger should provide hysteresis switch values of 1.875V and 3.125V. This prevents the ENDSTOP output from lingering in the digital logic grey area, eliminating spurious pulses during the transition between 0 and 1.

1.1.6 Capacitive Touch

Capacitive touch was selected as the interface for the system since it allows the buttons to be completely enclosed, preventing water ingress into the system. Since the primary objective of the system involves pots of water, this seemed prudent. The capacitive touch interface is based on the AT42 by Microchip Technology (D.3).

During part selection, the user interface was not yet fixed. This chip was selected primarily because it supported up to 11 key inputs, which provided a healthy margin for interface design. Another key consideration was the SPI interface provided on this chip. This allowed the sharing of the communications bus with the ADC with only one additional pin used as a chip select signal. This proved key as the final system design used every pin available on the AT89. The breakout board schematic is shown in Figure 1.7.

Normally the passive component values would be tested and adjusted with the electrodes designed for the system, but due to time constraints this was not possible. Values were chosen in the middle of the suggested range in the datasheet (D.3), with the option to modify the electrode design later if required.

Since the part is only available in a Quad Flat No-lead (QFN) package⁷, it was necessary to make a breakout board for the device, Figure 1.8. The board provides decoupling capacitors along with all necessary supporting passive components. All passive components are 0603 package, which allowed the board be miniaturized to just under 1.25" square.

⁵ Selected with reference to <http://www.ganssle.com/debouncing.htm>

⁶ Selected with reference to *The Art of Electronics* by Horowitz and Hill

⁷ There are no DIP capacitive touch chips that are not prohibitively expensive

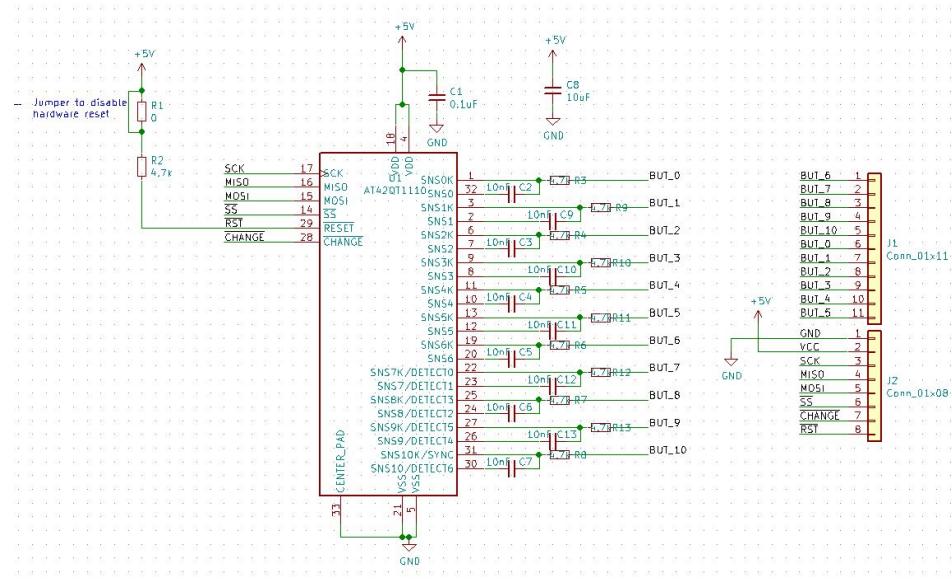


Figure 1.7: AT42 Capacitive Touch Breakout Schematic.

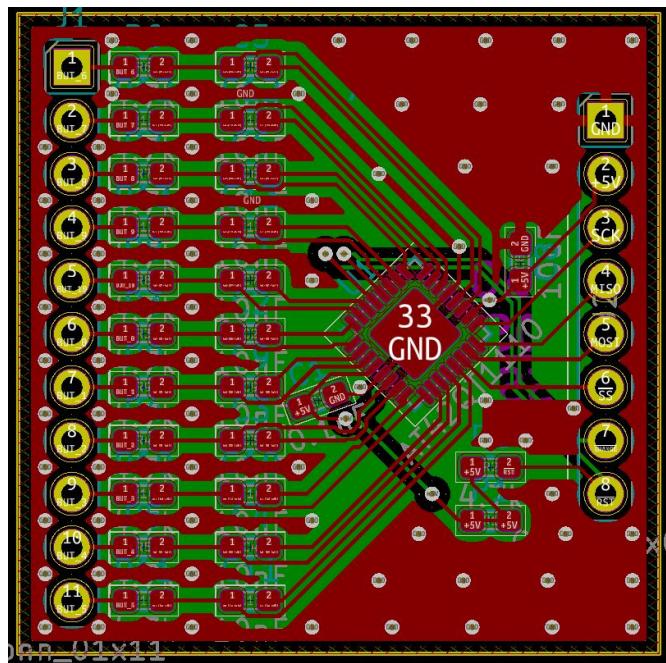


Figure 1.8: AT42 Capacitive Touch Breakout PCB.

1.2 FIRMWARE DESIGN

1.2.1 System Overview

The firmware design is centered around the **USER INTERFACE STATE MACHINE** and accompanying **PROGRAM STATE VARIABLE**. The **USER INTERFACE STATE MACHINE** controls the entire system, storing the current state in a globally accessible variable. Each sub-task is then able

to get the expected state from the **PROGRAM STATE VARIABLE** and act accordingly. The remaining structure is shown in Figure 1.9.

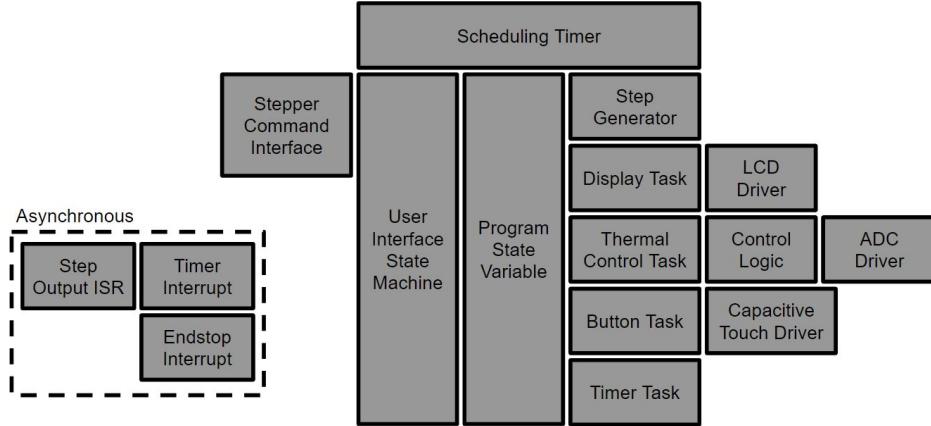


Figure 1.9: System Software Block Diagram.

Just over 1800 entirely new lines⁸ of code were written for this project. Each section below contains the count of new lines in that code module, but note that these were compiled by hand and may contain minor inaccuracies.

Using the Linux command line⁹, the total number of lines in the project was determined to be just over 3200 lines of code (including code reused from previous projects).

1.2.2 Scheduling Timer

All tasks, including the **USER INTERFACE STATE MACHINE** are scheduled by a timer. The timer runs continuously and dispatches each of the other tasks at set interval. This allows some tasks to be run at a higher frequency than others in the style of a cyclic executive.

The timer runs on a 5ms interval, and can dispatch tasks at 5ms, 10ms, 20ms, 40ms, and 80ms intervals. The tasks are scheduled at the following intervals:

- 5ms – STEP GENERATOR
- 10ms – BUTTON TASK
- 20ms – THERMAL CONTROL TASK, TIMER TASK
- 40ms – USER INTERFACE STATE MACHINE
- 80ms – DISPLAY TASK

180 new lines of code were written for this module.

⁸ Code not previously used in any other project

⁹ `find . -name '*' | xargs wc -l`

1.2.3 User Interface State Machine

The **USER INTERFACE STATE MACHINE** makes up the core logic of the system. It takes user input from the **BUTTON TASK** and monitors all the other tasks to determine the current and future state of the system. The state machine diagram is shown in Figure 1.10.

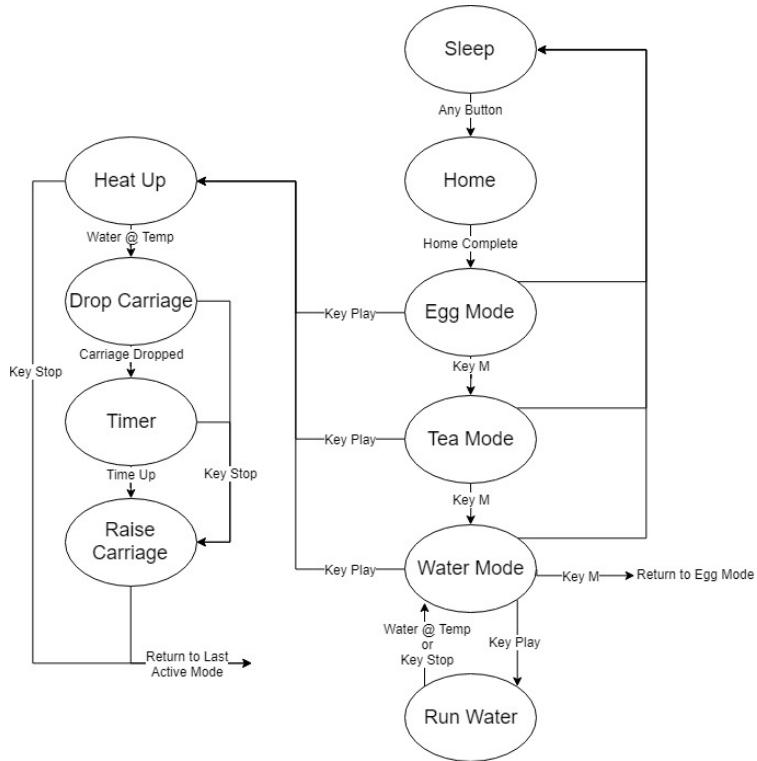


Figure 1.10: User Interface State Machine Diagram.

The **USER INTERFACE STATE MACHINE** interacts with the other tasks through the **PROGRAM STATE VARIABLE**, which takes the form of a struct that stores the state of the system including:

- Current temperature
- Goal temperature
- Temperature control enabled
- Current time
- Goal time
- Timer run enabled
- Preset name

- Current UI state
- Last UI mode
- Ring buffer of button events
- Preset updated flag

The first set of items are self explanatory for temperature and time-keeping. The second half bear a more detailed explanation. The *preset name* stores the name of the currently loaded preset¹⁰ for display by the LCD. The *last UI mode* stores the last mode the UI was in, so it can return to the same mode on completion or cancellation of the current cooking run. The *ring buffer* holds button events from the **BUTTON TASK** that have not yet been processed. The *preset updated flag* is used to alert the LCD when the user has selected a different preset. This allows the LCD to store minimal state, but also avoid updating the display more than necessary.

The **USER INTERFACE STATE MACHINE** can also trigger actions by the stepper motor using the **STEPPER COMMAND INTERFACE** to command the stepper motor to home itself or move the motor to a specific position. It should be noted that when the system goes into a sleep mode, the stepper position is no longer guaranteed, and as such the **USER INTERFACE STATE MACHINE** forces the stepper to be homed before any other tasks can execute.

278 new lines of code were written for this module.

1.2.4 Display Task & LCD Driver

The **DISPLAY TASK** maintains the **LCD** output to the user. In all states except sleep, it maintains an up to date temperature output. When flagged by the **USER INTERFACE STATE MACHINE** that the user preset has been updated, the **DISPLAY TASK** updates the **LCD** to the new preset name, target temperature, and target time. The **DISPLAY TASK** also monitors the state of the system timer, and when enabled, updates the current time output on the **LCD**.

The **DISPLAY TASK** sits atop the **LCD DRIVER**, which was written for a previous lab assignment in **ESD**.

157 new lines of code were written for this module.

1.2.5 Step Generator & Output Routine

The stepper motor driver is made up of three main parts: **STEPPER COMMAND INTERFACE**, **STEP GENERATOR**, and **STEP OUTPUT SERVICE ROUTINE**. The stepper driver was written with the intention to make it capable of full trapezoidal motion planning¹¹ for arbitrary movements.

¹⁰ Egg presets include very hard, hard, medium, and soft

¹¹ For more information on the trapezoidal motion profile see: <https://www.pmdcorp.com/resources/get/mathematics-of-motion-control-profiles-article>

The **STEP GENERATOR** is the core and most complex part of the stepper driver. The **STEP GENERATOR** takes commands from two ring buffers that specify a target position and velocity. Using these targets, it calculates step timings to accelerate from the current velocity to the target velocity. Once the target velocity is achieved, the **STEP GENERATOR** will maintain that velocity until the target position is achieved. Pulses are placed into a double buffer shared with the **STEP OUTPUT SERVICE ROUTINE** for execution.

The **STEP GENERATOR** also generates special control codes which are injected into the stream of step times to control the direction of the stepper motor. This is required since direction changes must be synchronized at specific points in the step pulse stream.

This driver did not achieve a fully generalized state however, since the user must command the **STEP GENERATOR** with an accelerating move and then a decelerating move. This simplifies the generator by lifting the requirement that the generator "look ahead" to determine when deceleration is required to start. Because the system only performs one specific, repeated motion, this reduction in complexity was acceptable.

Acceleration is implemented in a novel way to optimize the step generation process. Rather than use standard acceleration in $\frac{mm}{s^2}$, the system instead divides or multiplies the time in between steps by two during acceleration and deceleration respectively. In practice, this produces an exponential velocity curve which is actually smoother than a trapezoidal profile. This comes with the added benefit that acceleration calculations can now be greatly simplified by using bit shifts instead of multiplication and division.

The **STEP OUTPUT SERVICE ROUTINE** is where the pulses meet the motor so to speak. The routine uses a timer and the values provided by the **STEP GENERATOR** to execute step pulses at precision intervals, moving the motor. As previously mentioned, the routine also changes the direction of the motor on receipt of specific codes in the time stream. The use of double buffers allows the step timing to be generated asynchronously while the service routine executes them on precision intervals.

The final part of the stepper driver, the **STEPPER COMMAND INTERFACE**, provides a wrapped interface to add movement commands to the two ring buffers that serve as the input to the **STEP GENERATOR**.

562 new lines of code were written for this module.

1.2.6 Button Task & Capacitive Touch Driver

The **BUTTON TASK** periodically polls the state of the capacitive touch buttons and recognizes both short and long presses of the buttons. When a button event is registered, the **BUTTON TASK** then buffers it in the button event ring buffer for the **USER INTERFACE STATE MACHINE** to handle. Originally long presses were to be used for some other user interface functions like programming custom presets, but the scope of the

project proved too great to allow that to be implemented. Long presses are currently ignored by the state machine.

The **BUTTON TASK** sits on top of the **CAPACITIVE TOUCH DRIVER**, which provides initialization routines and routines to poll the current state of the **AT42**. the **CAPACITIVE TOUCH DRIVER** itself sits atop a **SPI** driver, which provides routines to complete basic transfers.

201 new lines of code were written for this module, not including 109 for the **SPI** driver which is shared with the **ADC** driver.

1.2.7 Thermal Control & ADC Driver

The **THERMAL CONTROL** task performs a modified bang-bang control of the kettle. When **THERMAL CONTROL** is activated, it calculates an intermediate temperature goal as 25% of the difference between the current temperature and the final goal temperature. The heater is then activated till this intermediate goal is reached, at which point it is disabled. The controller then waits while the kettle temperature overshoots and eventually stabilizes. At this point, the above procedure is repeated with a new intermediate temperature goal.

It can be seen that as the temperature gets nearer to the final goal temperature, the difference between the current temperature and the intermediate goal temperature gets smaller. This translates to shorter periods with the heater engaged, to less overshoot, and to more precise control. This has significantly reduced the overshoot over traditional bang-bang control, but there is still room for improvement.

THERMAL CONTROL sits atop the **ADC DRIVER**, which again utilizes the **SPI** driver. Similar to the **CAPACITIVE TOUCH DRIVER**, the **ADC DRIVER** implements initialization routines and routines to take readings from the **ADC**.

226 new lines of code were written for this module, not including 109 for the **SPI** driver which is shared with the **CAPCAITIVE TOUCH DRIVER**.

1.2.8 Timer Task

The **TIMER TASK** is one of the simplest tasks in the system. It simply maintains a Binary Coded Decimal (**BCD**) timer to be used to track the amount of time since cooking began.

43 new lines of code were written for this module.

1.2.9 Endstop Interrupt Service Routine

The **ENDSTOP INTERRUPT SERVICE ROUTINE** is triggered when the endstop is pressed. The routine simply updates a flag which the main routine can access, and stops the **STEP OUTPUT SERVICE ROUTINE** timer. Stopping this timer is done as an extra safety measure, since the timer

should never be running when the endstop is triggered. If the system enters an unforeseen state, this will prevent the motor from driving itself continually into the endstop.

62 new lines of code were written for this module.

1.3 MECHANICAL DESIGN

1.3.1 *Design Overview*

Mechanical design is not a focus of the [ESD](#) course, but as part of my goal to create a consumer quality project, time was spent on mechanical design. I feel that it should be at least briefly covered here.

The design of the system was undertaken using Onshape, a cloud based Mechanical Computer Aided Design ([MCAD](#)) tool. The primary structure of the device is made out of .118" clear acrylic. Acrylic was selected for its low cost, high clarity, and ease of laser cutting, which allows the parts to be fabricated quickly and precisely.

The interior of the base was designed with four partitioned areas. The primary purpose of these partitions was to limit paths for water ingress directly to the system electronics, and as such the opening for the belt and extrusion are in a single partition with no electronics. The secondary purpose of the partitioned areas is to limit the mingling of high and low voltage systems. One partition serves this purpose, containing the AC-DC converter and AC relay for controlling the kettle. A render of the complete system is shown in Figure [1.11](#).

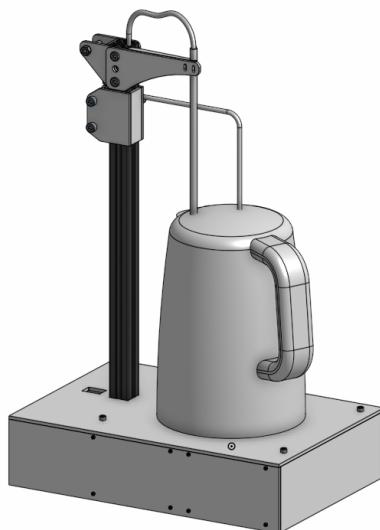


Figure 1.11: CAD Render of Eggbot created with Onshape [MCAD](#).

The other main component is the aluminum extrusion¹² which is the basis for the linear motion components. The motion platform eschews a traditional linear guide or roller bearings, and instead lets the acrylic carriage slide directly on the extrusion. Because the loading on the carriage is minimal and the accuracy requirement is non-existent, this solution is adequate. The carriage is moved by a GT2 belt¹³ running over a crowned pulley at one end and a 16 tooth cog at the other.

¹² <https://us.misumi-ec.com/>

¹³ <https://sdp-si.com/products/timing-belts/gt2.php>

2

IMPLEMENTATION & VERIFICATION

This chapter covers implementation and verification of the design outlined in Chapter 1. This includes details of the actual construction of the system, issues encountered, and tests performed to verify the system functionality. The second section also goes into further detail about how the firmware design was implemented and tested, including code listings.

2.1 HARDWARE IMPLEMENTATION & VERIFICATION

2.1.1 *System Overview*

The system was implemented in small chunks of concurrent design, firmware development, and hardware development. Usually this was limited to one block of the system hardware diagram (Figure 1.1) at a time. For example, the temperature sampling circuit was broken down into several elements: linearization, buffering, filtering, and ADC conversion. Each of these parts was implemented in hardware and tested before moving onto the next. Finally once the ADC was implemented, a basic firmware driver was implemented to test that portion of the circuit. After the basic driver was implemented, more complex functions are implemented referencing the requirements from the firmware block diagram (Figure 1.9). Finally these were tested. By using this method, each block in the diagram can be assured error free (mostly, at least) before proceeding to system level integration.

2.1.2 *Relay*

As mentioned in the design section for this module, 1.1.2, the relay is driven by a simple transistor arrangement with a flyback diode. I first populated the relay and flyback diode alone on a small piece of prototype PCB, and made long wire leads for the control inputs. This module was tested using a DC power supply, external to the board. Ideally I would have verified that the flyback diode was indeed clamping the inductive spike, but as my oscilloscope is analog, I did not have a way to capture this.

The module was then connected to the main PCB via a transistor already in place to drive an LED. Testing this revealed that the base driving resistor was not adequate to drive the load of the relay. A new base resistor value was then calculated as detailed in 1.1.2 and implemented by soldering a second resistor in parallel with the already installed resistor, as

removing the installed resistor was deemed too risky. This configuration worked.

2.1.3 Temperature Sensing

The system overview section (2.1.1) details the process taken with the temperature conversion circuit, so it will not be repeated here. This section will instead focus on the issues encountered in implementation. Originally, the circuit utilized a LM358 operational amplifier from Texas Instruments. When tested, the thermistor would produce expected values, but the voltage after the buffering and filtering circuit was not correct. Fortunately I had encountered this issue before, which made it easy to track down. The cause was the fact that the LM358 is not a rail-to-rail operational amplifier. Since the linearization range drives the thermistor across the majority of the 0-5v range, not using a rail-to-rail amplifier significantly modifies the signal.

After the issue was recognized, the amplifier was replaced with a MCP602 from Microchip (D.5). This part swap immediately remedied the issue. I also went ahead and ordered 25 to replace my supply of non rail-to-rail amplifiers, since the price is less painful than tearing my hair out.

The block was also tested with the kettle running on AC power just above it to check for induced noise. This showed very little noise on the line, but unfortunately I do not have an oscilloscope capture of this measurement.

2.1.4 Stepper Driver

The DRV8825¹ breakout board was soldered to a small piece of protoboard to allow the input and output pins to be broken out to connectors, and expose the microstepping control pins². This proved a boon since the original design called for the driver to be run in 1/16 step mode, which was eventually reduced to 1/4 step mode to reduce the load on the microcontroller. If these configuration pins had been jumpered with solder, this change would have been much more painful.

Limited testing was performed on the module itself since it was purchased complete. I did briefly install the driver in my 3D printer to verify that it was functional.

2.1.5 Endstop

The endstop circuit was plagued with the same rail-to-rail amplifier issues detailed in the section on temperature sensing. Fortunately this was

¹ <https://www.pololu.com/product/2133>

² The DRV can drive steppers in full step, 1/2 step, 1/4 step, 1/8 step, and 1/16 step based on the state of these jumper pins

quickly fixed by the same switch to a MCP602 from Microchip ([D.5](#)). Logic captures were performed to confirm the performance of the debouncer. Since it is directly connected to the processor interrupt pin, it is critical that it does not bounce.

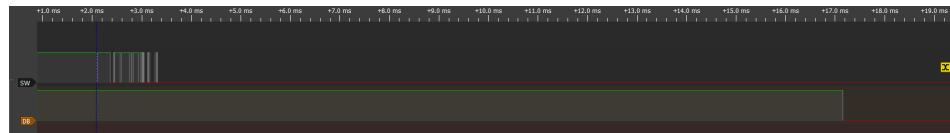


Figure 2.1: Capture showing the raw signal received from the button on top, and the debounced circuit output on the bottom. Captured with a generic Cypress FX2 based logic analyzer and sigrok/PulseView at 24MHz sample rate.

A logic capture of a button cycle is shown in Figure [2.1](#). The button is pressed just past 2ms in the capture, and bounces till about 3.5ms. The debounced output goes low at approximately 17ms. While 14ms is a relatively long latency between button press and output signal change, homing speed makes this a non-issue. Homing takes place at 20 mm/s, which means the carriage travels .3 mm (.010") in that latency time, which is not enough to hit the end of the endstop travel.

2.1.6 Capacitive Touch

As mentioned in the chapter about capacitive touch hardware design: [Capacitive Touch](#), I designed a small breakout board for the [AT42](#) capacitive touch chip from Microchip. My first attempt at designing a breakout used 0805 passives, and the components were fairly spaced out for the clearance I assumed I needed for hand soldering. This board design is shown in Figure [2.2](#).

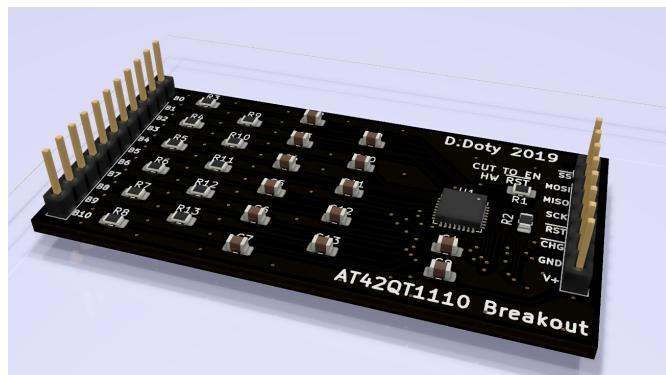


Figure 2.2: Capacitive touch breakout board for [AT42](#), version 1 design. Render created in KiCad EDA.

After printing out this board design at 1:1 scale and comparing it to my soldering iron, I decided that the original space allowance was larger than necessary and that I would also be comfortable attempting to move to smaller passive components, namely 0603's. This was also informed by the fact that I was committing to soldering a QFN32 in the [AT42](#), so moving to smaller passives did increase the total difficulty of the proposition by much.

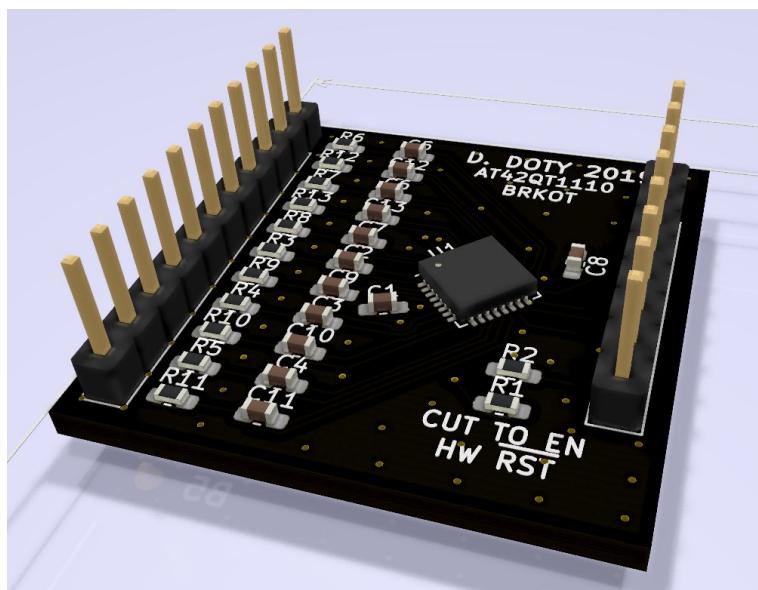


Figure 2.3: Capacitive touch breakout board for [AT42](#), version 2 design. Render created in KiCad EDA.

Version two of the design, shown in Figure 2.3 moves to 0603 passives, and much tighter component spacing as mentioned before. This allowed the board to be reduced to near 1.25" square. While this was not required for the design since board space is plentiful on the main [PCB](#), it was an interesting exercise nevertheless.

The capacitive touch [PCB](#) was ordered from [PCBWay](#). The boards arrived about a week after ordering them, in good shape. Due to the pricing model at [PCBWay](#), it makes sense to order a minimum quantity of 10. They often make a slightly larger quantity than ordered to offset process yield rates, but will send you the extra quantity if they all pass testing. Due to this, I received 11 boards instead of 10. I continuity tested all the pads of the boards after receipt and found no unexpected shorts or open connections.

I also ordered extra components to allow myself multiple attempts at populating boards. This proved unnecessary however, as the first populated board worked well. This board is shown in Figure 2.4.

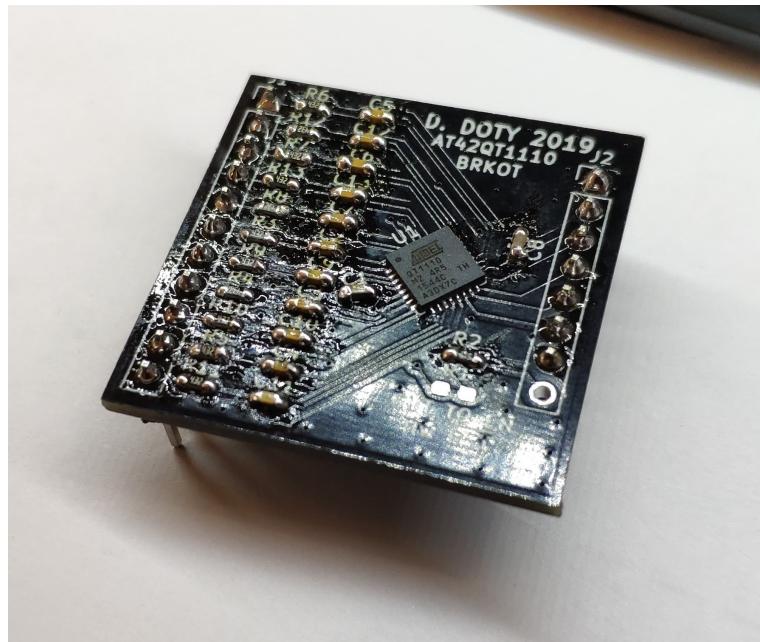


Figure 2.4: Capacitive touch breakout with all components populated, before flux removal.

The first board was tested using the [Bus Pirate](#), a general purpose USB adapter that supports various serial protocols over a command line interface to the computer. This allowed me to power the board, configure it, and test it directly from my computer without flashing a microcontroller to perform the exchange. Since the [AT42](#) supports EEPROM storage of configuration, this also allowed me to program all settings before the board was installed in the system.

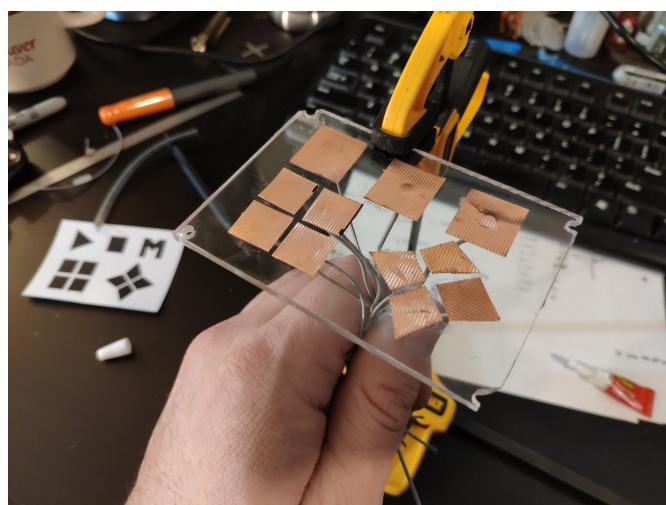


Figure 2.5: Capacitive touch electrodes.

The final part of the capacitive touch system is the electrode panel. The panel must have conductive pads wired to the input pins of the AT42. This was achieved with thin copper foil, as shown in Figure 2.5. The copper foil was cut to roughly the size of the desired buttons and a wire was soldered to the back of each electrode near the center. Then a small square of acrylic had holes drilled in it for the electrode wires to pass through, and the electrodes were glued to the surface. The wire holes in the back of the acrylic were then potted with hot glue as a strain relief. The finishing touch was gluing a paper sheet with the graphical buttons over the copper foil.

Overall the electrodes work very well, but are somewhat less sensitive than desired when placed behind the acrylic chassis. This could be corrected by making the button electrodes larger, and also by modifying the sense capacitors on the breakout board. As mentioned in the previous section, [Capacitive Touch](#), the sense capacitors were selected based on the recommended values in the datasheet due to time constraints.

2.2 FIRMWARE IMPLEMENTATION & VERIFICATION

2.2.1 System Overview

While most base level driver functionality was implemented in parallel with the hardware modules, the top level firmware design is a more all-or-nothing approach. Stages of development and testing were divided as possible, but due to the interdependencies of an integrated system these chunks of development were quite large.

2.2.2 Scheduling Timer

The scheduler, shown in Appendix C.3, was implemented using timer set to a period of 5ms, and a counter that is incremented in the ISR and used to determine when to dispatch services, using a bit addressable flag in the AT89.

Timer 2 was used in 16 bit auto-reload mode set for an interval of 5ms. At each 5ms interval, a counter variable is incremented in the ISR, and XOR'd with its previous value. For those familiar with binary counting, each bit changes at a different frequency (e.g. the LSB changes at every increment, the LSB+1 changes at every other increment, LSB+2 at every 4th increment and so on). By XORing the count with itself, only the bits that change are returned. Each of these bits can then be OR'd with a bit addressable flag (to prevent clearing a flag that was already set), creating a set of flags that are set periodically at different multiples of the base frequency. A portion of this code is shown in Listing 2.1. This is repeated through 8 bits to dispatch all 8 flags.

Listing 2.1: Snippet of Scheduler ISR

```

1 // XOR the count with the previous count
2 xrl    a,r7
3
4 // Set dispatch bits
5 setb _task_flag_0      // Task 0 gets set every ISR run
6 rrc A                  // Just waste bit 0 since it
7           changes every time (for task 0)
8
9 rrc A                  // Rotate right into carry (
10    b0 -> carry)
11 orl C,_task_flag_1     // Or the flags together (ensure you
12           don't clear something already set)
13 mov _task_flag_1,C     // Set the flag
14 rrc A
15 orl C,_task_flag_2
16 mov _task_flag_2,C
17 rrc A

```

Since the scheduler runs more frequently than any other piece of code in the system (except the stepper [ISR](#), addressed later), it was optimized in assembly to reduce the load on the system. The original C version of the [ISR](#) compiled to approximately 120 assembly instructions not including standard [ISR](#) housekeeping instructions. The optimized routine takes just 20 assembly instructions, a reduction of over 80%.

The dispatch timing was verified using a helpful trick in the Linux command line:

```
cat /dev/tttx | ts -i -m %.s
```

This command shows the output from serial device tttx (where x should be a number) with a timestamp of the number of seconds elapsed since the last newline character was printed. This is especially useful for timing code execution on a microcontroller by placing a *putchar*(n) at each point where time should be printed. Since *putchar* is generally a very lightweight function, this allows for timing of execution with minimal interference with the execution of the program as long as measurement points are far enough apart to prevent blocking while waiting for the last transmission to complete.

The scheduler in main takes the form of a series of if else if statements, a few of which are shown in snippet 2.2. Else if's are used to implement a pseudo priority order between the service, which follows Rate Monotonic³ scheduling policy. While the scheduler is not preemptive, this does mean that the highest priority task ready to run when the system is idle will get to run. Note that these if statements are compiled as an atomic flag check and clear, which avoids the need for a critical section.

Listing 2.2: Snippet of Main Scheduler

³ Rate Monotonic specifies that the highest frequency task gets the highest priority

```

1      if(task_flag_0)           // 1x sched_period
2      {
3          task_flag_0 = 0;
4          // Calculates step timing for the stepper
5          stepper_sequence_generator();
6      }
7      else if(task_flag_1)     // 2x sched_period
8      {
9          task_flag_1 = 0;
10         // Reads button states and puts them in the
11         // buffer
12         task_buttons();
13     }

```

2.2.3 User Interface State Machine

The design of the user interface state machine was detailed thoroughly in Section 1.2.3, so that will not be repeated here. A brief pseudocode version of the service is offered below in snippet 2.3.

Listing 2.3: User interface pseudo code.

```

1 ring_remove(button_events)
2 if(no_button_events)
3 {
4     idle_counter++           // increment an idle
5     counter
6     if(idle_counter)
7     {
8         system_state = sleep; // Go to sleep if idle for
9         // too long
10    }
11 }
12 switch(system_state)
13 {
14     case system_states:
15     // Implementation of state machine
16 }

```

The full code implementation of the user interface state machine can be seen in Appendix C.2, lines 463-629. Due to the nature of this module, there is not much that can be done in the way of block level verification, so verification of this module was performed during system level testing.

2.2.4 Display Task & LCD Driver

The first implementation of the display task simply updated all text on the [LCD](#) at every dispatch based on the values present in the system state variable. As system level testing started to reveal that the system was overloaded, this was revised to mostly eliminate [LCD](#) updates except where the values on the screen need to change.

This was achieved through a flag from the user interface task, which notifies the display task when the user changes the preset, requiring the update of preset names and values. The display task also monitors the system state to determine when the timer value needs to be updated and when it can be left static.

The display task utilizes an [LCD](#) driver that was written for the course on a previous assignment, so it will not be detailed here. The display task code can be seen in Appendix [C.2](#) lines 354-460.

Another part of the optimization of the display task was the creation of a custom string formatting function, called `micro_sprintf`. This is an assembly optimized `sprintf` that only supports the conversion of unsigned 8 bit numbers to ASCII characters. This function can be seen in Appendix [C.2](#) lines 741-790.

The optimization efforts on the display task led to an overall execution time reduction from 32ms to execution times ranging from 15ms down to .07ms⁴.

2.2.5 Step Generator & Output Routine

Section [1.2.5](#) details the design of the stepper system. From the design, one can tell that the module was originally intended to be non-blocking, allowing other tasks to execute in the times in between step planning. In practice, this did not prove feasible due to overloading the system. During system level testing, the stepper would often stop when reaching a high speed as the [ISR](#) catches the step generator and runs out of steps to execute. This was remedied with four changes: heavily optimizing the stepper [ISR](#), reducing the number of steps per millimeter⁵, decreasing the speed⁶, and making the step generation routine blocking. The step generation routine now takes control of the processor and executes alone until all requested movements are completed.

This module was by far the most difficult and time consuming to write. Fortunately, it could be tested in seclusion without the other modules. Figure [2.6](#) shows steps being captured for a short positive move, and then a short negative move. While the details are not easy to see in the image,

⁴ Execution time is much more variable now since the whole display is no longer updated. The worst case of 15ms happens only when changing presets

⁵ Changing the [DRV](#) microstepping setting from 1/16 to 1/4

⁶ Also beneficial in reducing the noise the system makes, which was considerable at high speed

it is possible to see the change in the pulse density while the stepper accelerates and then decelerates.

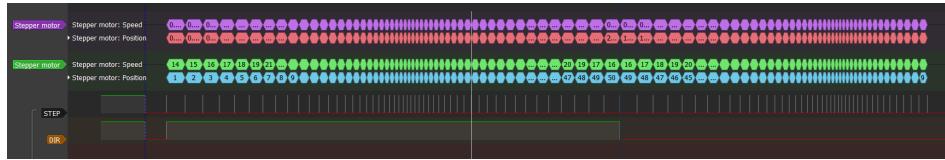


Figure 2.6: Step generation capture. Captured with a generic Cypress FX2 logic analyzer and sigrok/PulseView at 24MHz.

Figure 2.7 shows data captured from a short low speed movement and plotted to determine velocity throughout the move. Here the exponential velocity profile can be seen which results from using time division based acceleration⁷. This profile is actually desirable since it reduces the instantaneous acceleration experienced by the motor (and egg).

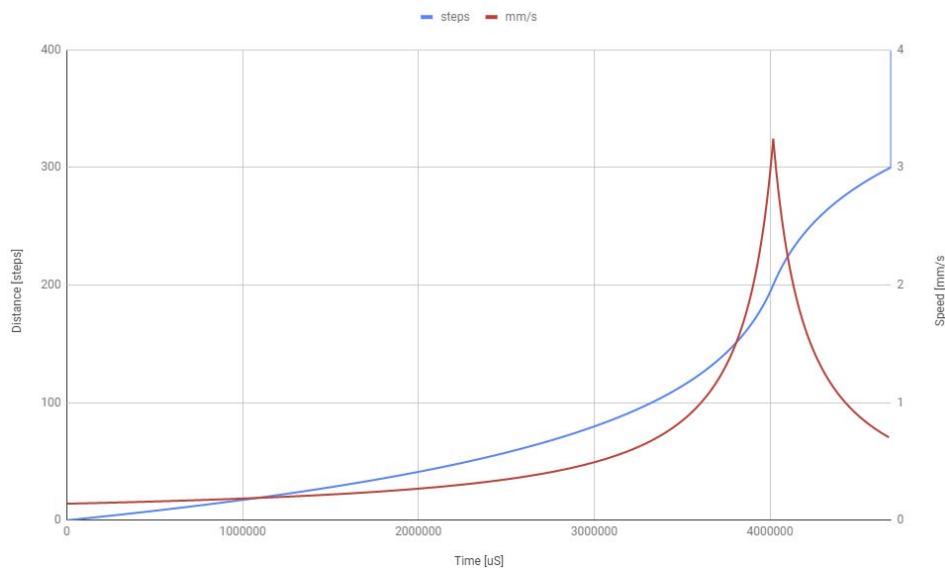


Figure 2.7: Plot of stepper movement showing velocity profile and distance. Note that the capture is truncated before the move is complete.

The stepper ISR was another area of intense assembly level optimization. The first iteration used a ring buffer in XDATA memory to store the step times, and was completely written in C. This version compiled to 453 assembly instructions. Given a max step rate of 2500⁸ steps per second, this ISR needs to execute 1.1 million instructions per second to keep up, not

⁷ Discussed in Section 1.2.5

⁸ 25 steps/mm, 100mm/s

even including the step generation routine. The [AT89](#) running at 11.0592 MHz can only execute 921 thousand instructions per second⁹, meaning we've already overrun. At this point the [ISR](#) was thoroughly optimized, switching to a double buffer in directly addressable memory rather than the ring buffer in XDATA, and rewriting the routine in assembly. The final routine is just 52 instructions total, and most paths through the [ISR](#) are significantly less than that. This equates to a reduction of 88%.

With the new [ISR](#), achieving the step rate of 2500 steps/second takes just 130 thousand instructions per second, leaving 85% of the processor time to complete other tasks, like step planning. The stepper code is available in Appendix [C.6](#).

2.2.6 Button Task & Capacitive Touch Driver

The button task and capacitive touch driver are relatively simple. The driver provides wrappers for the two operations required: configuring the board and polling the buttons states as seen in Snippet [2.4](#). One interesting thing to note that was discovered during test, the capacitive touch chip was not storing its configuration in EEPROM. It is not clear if this was due to an error in the routines used to program the EEPROM, or if this was a defect with the part. To resolve this, the critical settings are reprogrammed at every system boot. The full code is available in Appendix [C.11](#).

Listing 2.4: Snippet of Capacitive Touch Driver.

```

1 // Configure the captouch IC AT42QT1110, verify status is good
2 void captouch_configure(void);
3
4 // Get the status of all buttons. If there is a failure this will
   return 0
5 button_t captouch_poll_buttons(void);

```

The button task uses a basic state machine to detect the end of a button press and report it into the button event ring buffer. The end of the button press is reported rather than the start because it allows detection of long presses. The original user interface concept called for some extra functions that would be accessible through long presses, but these were not implemented due to schedule restrictions. The full button task is available in Appendix [C.2](#) lines 191-242.

2.2.7 Thermal Control & ADC Driver

The temperature readings are taken by the [ADC](#) driver, and converted to Fahrenheit by the thermistor functions. [ADC](#) readings are separated from the thermal control task because in an earlier version of the code, the [ADC](#) was oversampled. This proved unnecessary however, because of the

⁹ All assuming single cycle instructions

very long time constant of the system and heavy filtering of the thermistor voltage.

Temperature calibration was verified by measuring the temperature of boiling water and that of iced water. While a two point calibration is not ideal, no high resolution thermometer was available at the time to perform a more thorough calibration. The full code for the [ADC](#) driver and thermistor conversion functions can be viewed in Appendix [C.13](#) and [C.15](#) respectively.

Thermal control, as mentioned in Section [1.2.7](#), is based on a modified bang-bang controller. This controller was implemented to attempt to reduce setpoint overshoot observed with a traditional bang-bang controller, while avoiding cycling the relay too often. A data capture for a modified bang-bang run is shown in Figure [2.8](#).

This plot shows the extreme amount of overshoot in the system, most likely due to a large thermal mass in the heater relative to the mass of the water, and poor water circulation. The modified bang-bang control is adequate in reducing this overshoot for the time being however. The full thermal control code can be seen in Appendix [C.2](#) 288-352.

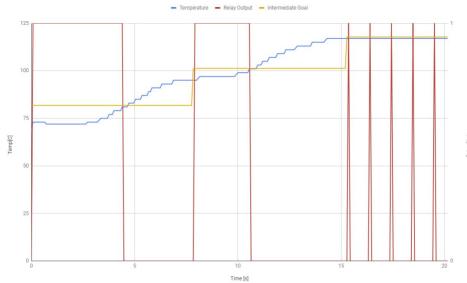


Figure 2.8: Plot of temperature and relay output for a heating run with a target temperature of 120 °F.

2.2.8 Timer Task

The timer task is very simple. At every dispatch it checks if the timer is enabled, and, if so, adds time to the current time value. This was optimized further at the assembly level to make use of the [BCD](#) instructions available. The [BCD](#) adjust instruction drives the design of the timer to count up, rather than down, since the instruction only supports addition. Unfortunately no side by side profiling of this function and its original C implementation was performed. The full code for the Timer is available in Appendix [C.2](#) lines 244-286.

2.2.9 Endstop Interrupt Service Routine

The endstop is possibly the simplest of all the software modules. It simple raises a flag when the endstop is triggered. Later, an additional line was added to stop the step generation timer. This stops the stepper motor if the endstop is hit unexpectedly. The full code can be seen in Appendix C.7.

2.3 SYSTEM LEVEL VERIFICATION

Hardware assembly was a daunting task, as all the components had to be assembled without inadvertently breaking something after every component had been verified. Assembly was completed one module at a time, interspersed with quick tests I had created earlier in development. This allowed me to confirm each component in the system was functioning successfully before moving on to the next one. System level issues are notoriously difficult to find, so this approach helped limit surprises.

System level firmware verification was a challenging task, but eased significantly by the unit level tests performed during the rest of development. Task execution was verified using individual putchar() commands in each section of the scheduler. This allowed me to quickly profile which tasks were running and how often, and was instrumental in diagnosing the stepper stall issues that plagued early integration.

One major issue cropped up late in implementation that was very difficult to diagnose. Just before I was due to demonstrate the system, I unplugged the programmer from the board and installed the bottom of the enclosure. Suddenly, the capacitive touch pad became unresponsive and registered random touches when not being touched. After reconnecting the programming cable to the system for debugging, the problem disappeared. Eventually I discovered that the problem is triggered by disconnecting the system from earth ground. When the programmer is connected, the system is grounded through the computer. When it is disconnected, the system is transformer isolated by the power supply. While the cause of this issue has not yet been identified, I hope to continue to work on it after the course is complete.

3

CONCLUSION

3.1 LESSONS LEARNED

Below is a list of my favorite lessons learned through the course of the project

INCREMENTAL DEVELOPMENT Breaking the project up into modules from the hardware and software block diagram was invaluable. For every untested module added, the number of potential issues in a system goes up exponentially, and system level issues are much more difficult to find than module level issues. By completing each module and testing it before integrating it, it is much less likely that system level issues will exist. If issues do arise, already available working known, working tests can be run on the integrated system to help pinpoint that issue.

INCREMENTAL INTEGRATION By the same logic as above, do not add all the parts to the final system at the same time, even if they have been individually tested. Instead, add modules to the system one by one, testing the test after each new addition. This may seem like a waste of time, but it only takes one difficult issue to more than offset all the time saved by skipping tests.

CONNECTORS Every cable that goes between two PCBs should have a connector on it. These are clearly boundaries between two modules, and the ability to disconnect them will greatly ease debugging.

DON'T WAIT TO DEBUG Trying to guess the source of a problem and changing random things, instead taking out the oscilloscope, is a waste of time. Use the debugging tools available immediately, and keep systematic notes of what is changed while debugging. You never know what hasty change might make or break the system, and having the notes to backtrack changes to a working system is valuable.

STREAMLINE BUILD PROCESS Spend the time up front to make a nice build environment for yourself. Create automation scripts, write a nice makefile, explore parallel compilation. You will be compiling and flashing code potentially hundreds of times. Make it painless.

SDCC SDCC is not perfect and often leaves a lot of room for assembly level optimization. Do not be intimidated by assembly level optimization.

PROCRASTINATION Do not wait to start. It is very rare that a task takes less time than expected.

3.2 FUTURE DEVELOPMENTS

Here are a few areas of potential improvement in the project's future.

STEPPER OPTIMIZATION Improve the stepper planning routine and make it cooperative again. Currently it blocks execution, which can sometimes leave the relay on for a few seconds longer than intended driving the system far off the temperature set point.

CODE OPTIMIZATION Profile the system code, and optimize the most frequently called functions, likely at the assembly level. The system currently bogs down when heavily loaded, leaving the user interface with noticeable latency. Ideally this would be completely eliminated.

THERMAL CONTROL Improve the thermal control algorithm, most likely using a predictive heating model. By measuring the response of the system to a known length heat input pulse, I think I can estimate the amount of water present in the kettle. Using this, it should be possible to predict how long a heat input pulse is required to heat the water to the set point. This could greatly reduce overshoot without requiring the relay to cycle frequently.

CAPACITIVE TOUCH Find the root cause of the capacitive touch earth ground issue and resolve it. Not a lot to say for this one.

INTERFACE EXTENSIONS Extend the user interface to allow the programming of custom preset times and temperatures. These can be stored in the on board EEPROM to preserve them through power cycles.

MECHANICAL IMPROVEMENTS The acrylic case could be thickened and reinforced to make the system more robust for daily use.

3.3 CONCLUSION

This project covered the creation of a single axis, controlled temperature bath, egg cooking robot from design, through implementation, and testing. Several notable issues were encountered through the course of the project, namely system overload due to the stepper pulse output routine, thermal control overshoot, and capacitive touch issues. Overall the project was a success, resulting in a system that can cook eggs and make tea, albeit noisily.

4

ACKNOWLEDGMENTS

I'd like to thank my girlfriend, Cathy Gibson, for allowing me to use her as an excuse to build a ridiculous egg cooking robot, supporting me throughout the process, and proofreading this paper. I know you love listening to me rant about interrupt service routines, and don't worry because there is more to come.

I'd also like to thank Professor Linden McClure, for whose class this project was created. I appreciate the chance to follow a silly idea to its end.

One more thank you for the teaching assistants for [ESD](#), for their help and guidance throughout the semester.

A big thanks to André Miede, who created this \LaTeX template. It is licensed under the [GNU GPL](#) and used with permission.

<https://bitbucket.org/amiede/classicthesis/>

A

BILL OF MATERIALS

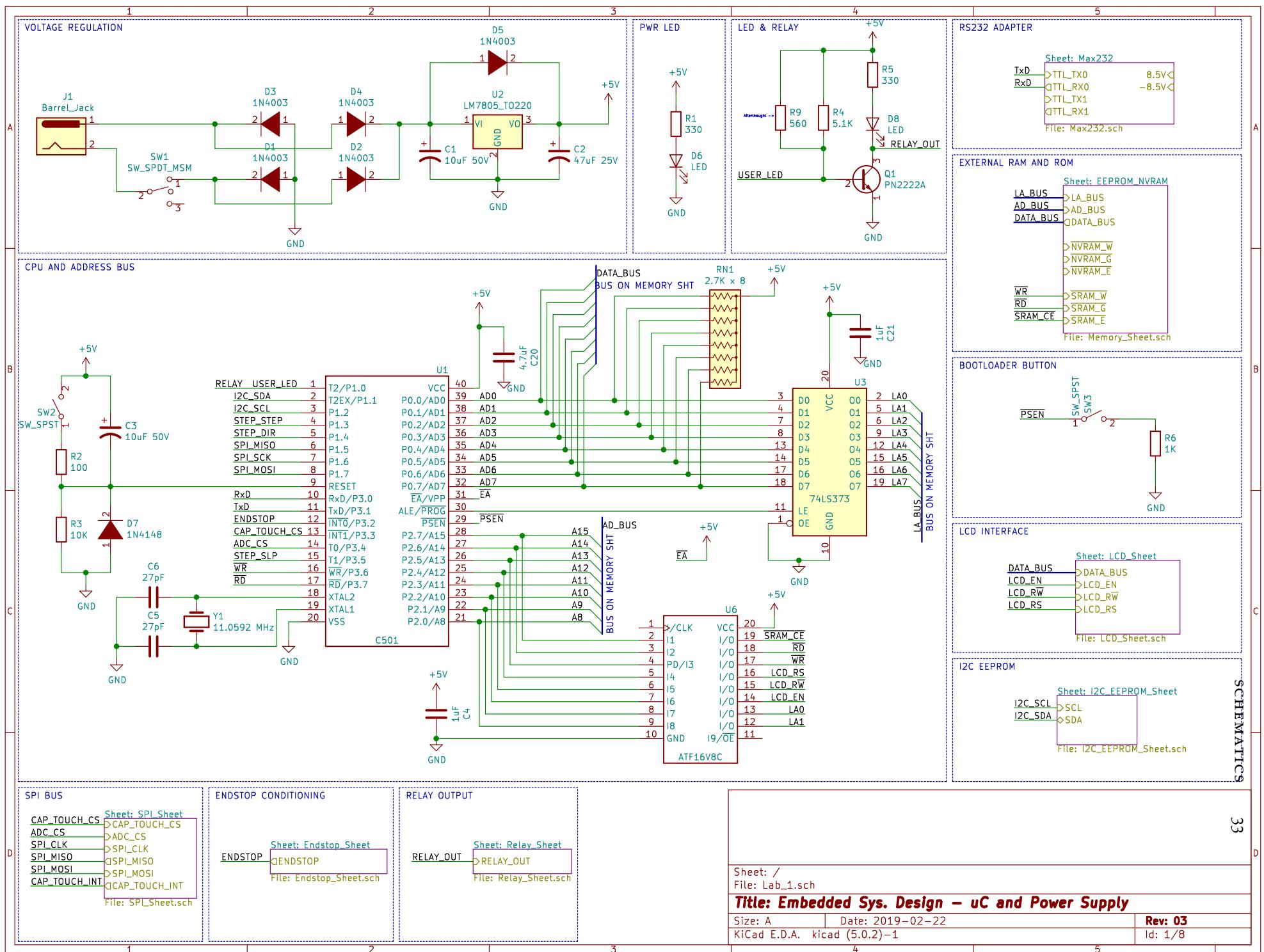
The table below contains all the items I purchased for this project, and some that I already had. Note that this does not reflect how much I paid, but common prices I was able to find online. The list also does not include some miscellaneous items like fasteners and bearings, which I already had on hand and do not have a good price estimate for.

NAME	PART NUMBER	QUANTITY	PRICE	EXT.
Touch IC	AT42QT1110	1	.48	.48
Touch Passive	10nF X7R Capacitor	11	.034	.374
Touch Passive	2.2k Resistor	11	.10	1.10
Endstop	SW1982-ND	1	.79	.79
ADC	MCP3002-I/P	1	1.84	1.84
Touch PCB	N/A	1	7.00	7.00
Op Amp	MCP602-I/P	2	.48	.96
Relay	PB3351-ND	1	2.95	2.95
Flyback Diode	1N4007	1	.11	.11
SRAM	1450-1480-ND	1	3.20	3.20
Stepper Driver	DRV8825	1	8.95	8.95
Laser Cut Chassis	N/A	1	61.52	61.52
Magnets	N/A	1	.0899	.0899
Thermistor	N/A	1	14.99	14.99
Kettle	N/A	1	20.99	20.99
Stepper Motor	N/A	1	10.99	10.99

Table A.1: Parts, costs, and quantities.

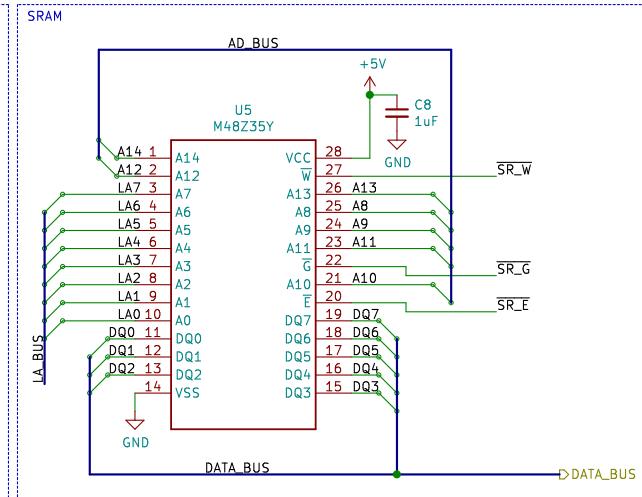
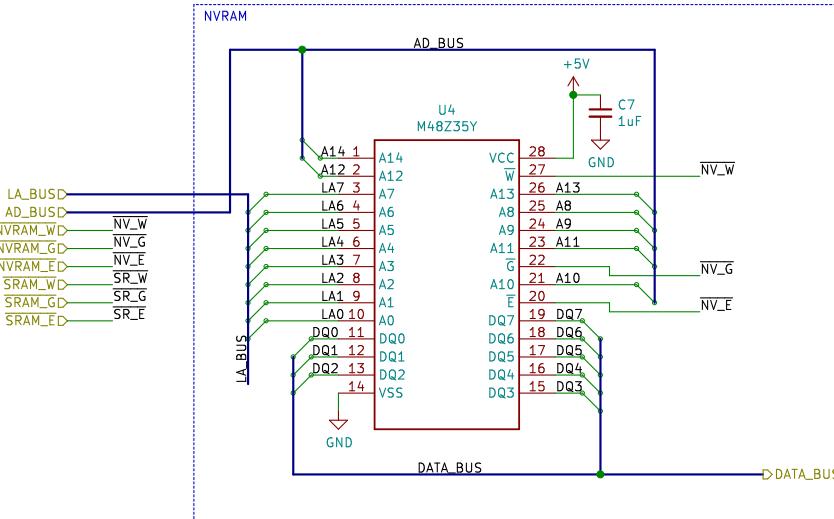
B

SCHEMATICS



1 2 3 4 5 6

A



Sheet: /EEPROM_NVRAM/
File: Memory_Sheet.sch

Title: Embedded Sys. Design – Memory Interface

Size: A4 Date: 2019-02-15
KiCad E.D.A. kicad (5.0.2)-1

Rev: 02
Id: 2/8

1

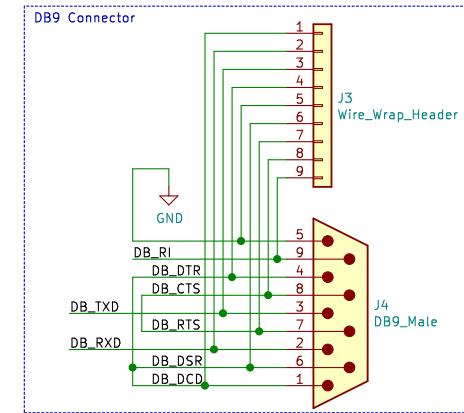
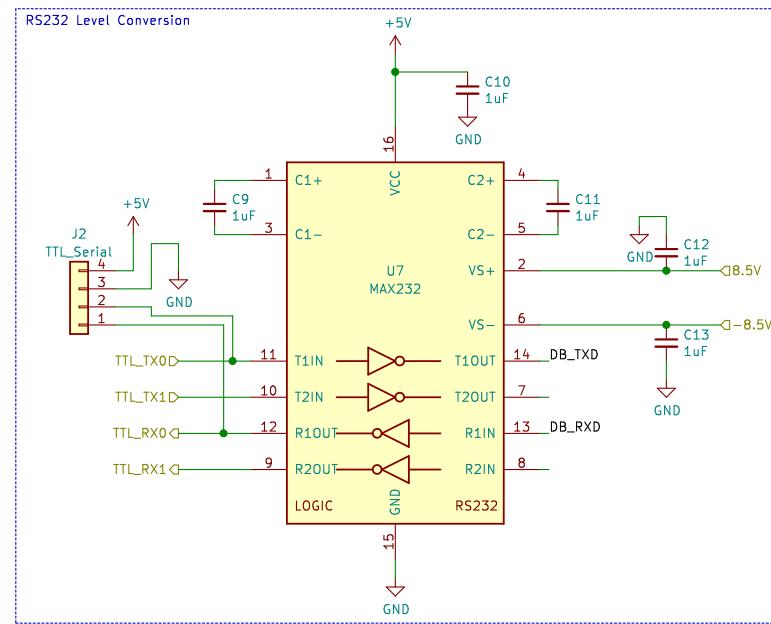
2

3

4

5

6



Sheet: /Max232/
File: Max232.sch

Title: Embedded Sys. Design – RS232 Serial

Size: A4 | Date: 2019-02-22
KiCad E.D.A. kicad (5.0.2)-1

Rev: 01 | Id: 3/8

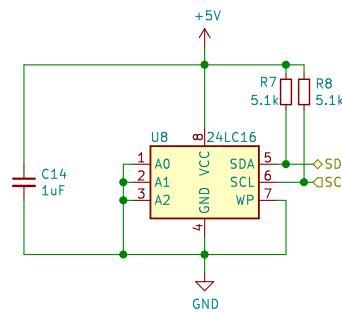
1 2 3 4 5 6

A

B

C

D



SCHEMATICS

36

Sheet: /I2C_EEPROM_Sheet/
File: I2C_EEPROM_Sheet.sch

Title: I2C EEPROM Interface Sheet

Size: A4 Date: 2019-03-29
KiCad E.D.A. kicad (5.0.2)-1

Rev:
Id: 4/8

1 2 3 4 5 6

1 2 3 4 5 6

A

A

B

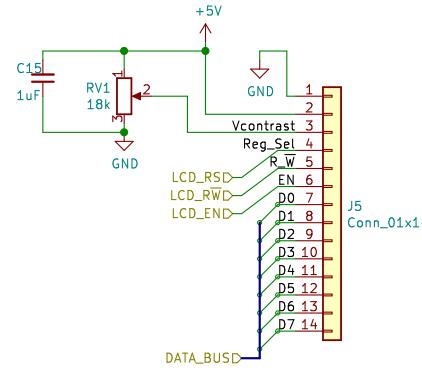
B

C

C

D

D

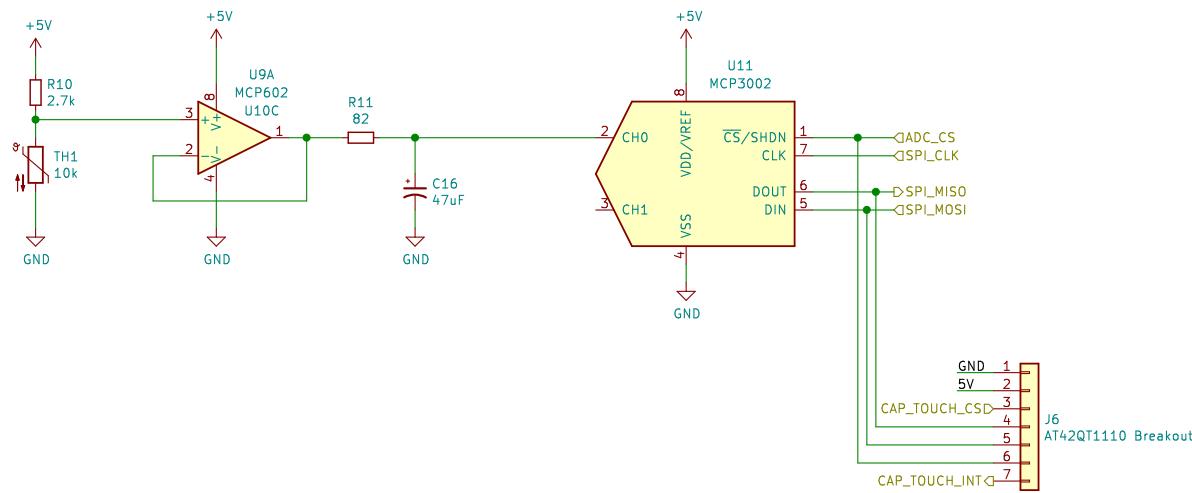


Sheet: /LCD_Sheet/
File: LCD_Sheet.sch

Title:

Size: A4	Date:
KiCad E.D.A. kicad (5.0.2)-1	Id: 5/8

1 2 3 4 5 6

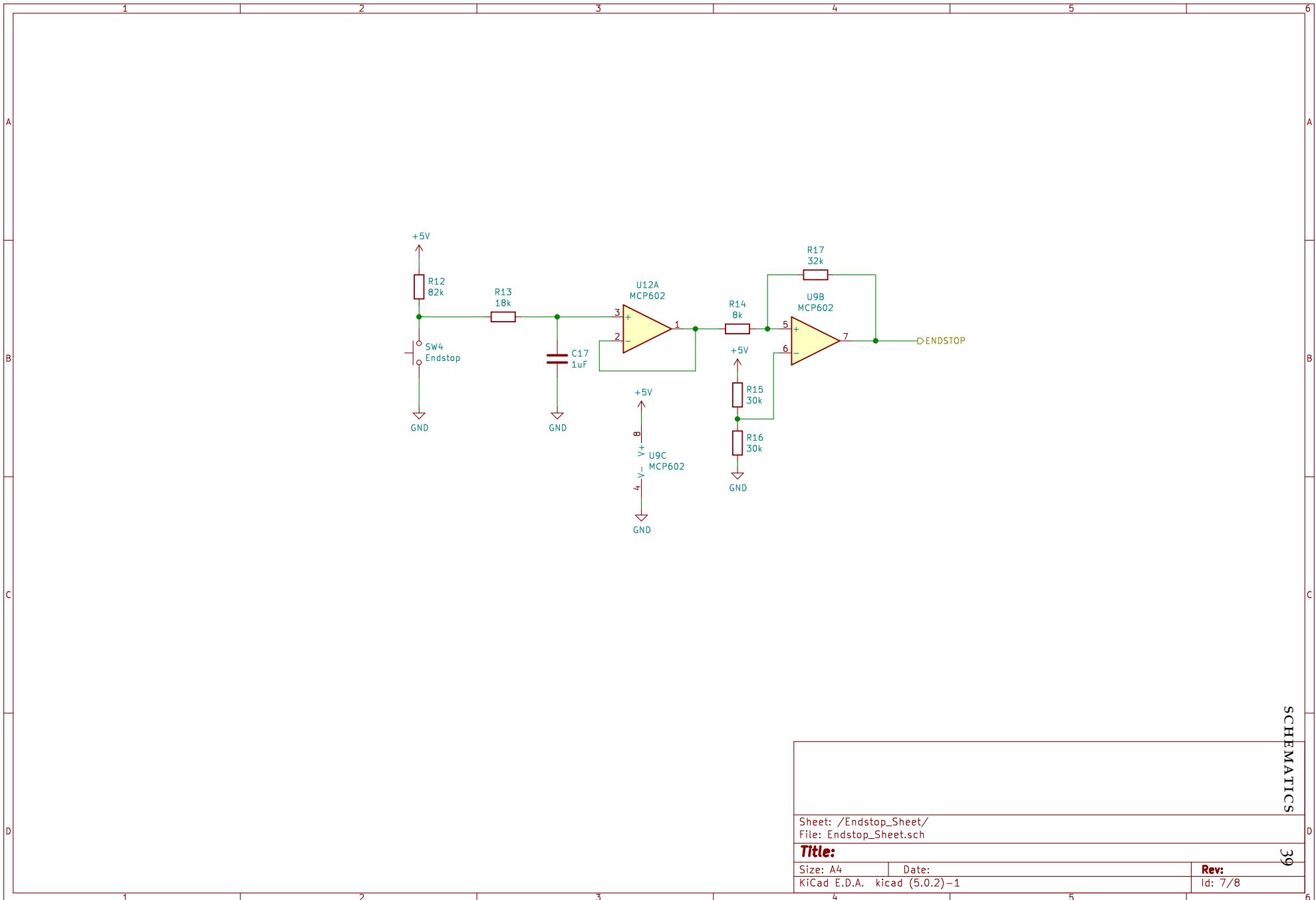


Sheet: /SPI_Sheet/
File: SPI_Sheet.sch

Title:

Size: A4 | Date:
KiCad E.D.A. kicad (5.0.2)-1

Rev: 6/8



A

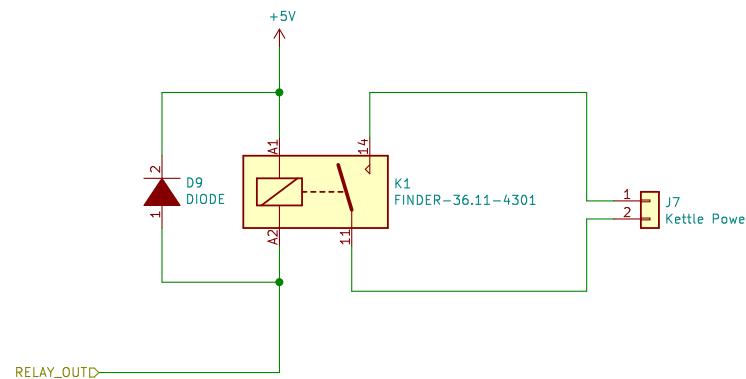
B

C

D

SCHEMATICS

40



Sheet: /Relay_Sheet/
File: Relay_Sheet.sch

Title:

Size: A4 | Date:
KiCad E.D.A. kicad (5.0.2)-1

Rev:
Id: 8/8

C

CODE

This appendix includes only the new code written for this project. See the [table of contents](#) for links to each file.

Listing C.1: Pin Settings Header

```
1  /* ===== PROGRAM INFO ===== */
2  // Dominic Doty
3  // ECEN 5613
4  // Final Project
5
6
7
8
9  /* ===== PIN MAPPING DEFINES ===== */
10
11 #define RELAY_OUT          P1_0
12 #define I2C_SDA            P1_1
13 #define I2C_SCL            P1_2
14 #define STEP_STEP           P1_3
15 #define STEP_DIR            P1_4
16 #define SPI_MISO            P1_5
17 #define SPI_SCK             P1_6
18 #define SPI_MOSI            P1_7
19
20 #define UART_RX             P3_0
21 #define UART_TX             P3_1
22 #define ENDSTOP             P3_2
23 #define CS_CAPT              P3_3
24 #define CS_ADC               P3_4
25 #define STEP_SLP              P3_5
```

Listing C.2: Main Source

```

1  /* ===== PROGRAM INFO ===== */
2 // Dominic Doty
3 // ECEN 5613
4 // Final Project
5
6
7 #include <sdcc-lib.h>
8 #include "mcs51/at89c51ed2.h"
9 #include <string.h>
10
11 #define DEBUG_PRINT
12
13 #include "pin_map.h"
14 #include "serial.h"
15 #include "i2c.h"
16 #include "lcd_driver.h"
17 #include "adc_driver.h"
18 #include "thermistor.h"
19 #include "endstop.h"
20 #include "captouch_driver.h"
21 #include "delay.h"
22 #include "stepper.h"
23 #include "scheduler.h"
24
25
26 /* DEFINES */
27 #define AUXR_1024RAM_EXTRAM      0x0C
28 #define RELAY_OFF                0
29 #define RELAY_ON                 1
30
31 /* Intertask Data */
32 typedef enum{
33     sleep,
34     home,
35     egg,
36     tea,
37     water,
38     run_heatup,
39     run_dropcar,
40     run_time,
41     run_raisecar,
42     run_water
43 }ui_state_t;
44
45 typedef struct
46 {
47     uint8_t temp_current;    // current temp in F
48     uint8_t temp_goal;       // target temp in F
49     bool temp_enable;        // enable temperature control
50     uint8_t time_current[3]; // BCD duration timer MM:SS.SS

```

```
51     uint8_t* time_goal;           // BCD cooking time
52     bool time_enable;           // Enable the timer
53     char* preset_name;          // Name of the loaded preset
54     ui_state_t state;           // Current UI state
55     ui_state_t restore_state; // Last active state before running
56     program
56     ring_32_t button_events; // Holds events sent from captouch
57     bool preset_updated;    // Flags if the preset choice or
58     number has been updated by ui
58     uint8_t cursor_location; // Current location of the LCD cursor
59 }ui_situation_t;
60
61 ui_situation_t system_state;
62
63
64 /* PRESETS */
65 typedef struct
66 {
67     char name[10];
68     uint8_t temp;
69     uint8_t time[2];
70 }preset_t;
71
72 typedef struct
73 {
74     preset_t SQ_UL;
75     preset_t SQ_UR;
76     preset_t SQ_LL;
77     preset_t SQ_LR;
78 }mode_presets_t;
79
80 mode_presets_t egg_preset;
81 mode_presets_t tea_preset;
82 mode_presets_t water_preset;
83
84
85 /* TASK FUNCTIONS */
86 void task_buttons(void);
87 void task_timer(void) __naked;
88 void task_adc_samp(void);
89 void task_therm_control(void);
90 void task_display(void);
91 void task_ui(void);
92 void select_preset(button_t button, mode_presets_t* mode_preset);
93 void presets_initialize(void);
94 void micro_sprintf(__far char* destination, uint8_t number) __naked;
95
96
97 /* MAIN */
98 void main( void )
99 {
100     // Turn off the kettle just in case
```

```

101     RELAY_OUT = RELAY_OFF;
102
103     // Initialize the Presets
104     presets_initialize();
105
106     // Communications Init
107     serial_configure(BAUD_57600);
108     spi_configure(spi_ss_disable,
109                   spi_mode_master,
110                   spi_cpol_idle_high,
111                   spi_cpha_sample_return_idle,
112                   spi_baud_div8);
113
114     // Stepper System Init
115     stepper_configure();
116     endstop_configure();
117     stepper_sleep();
118
119     // LCD Init
120     lcd_configure();
121
122     // Peripherals Init
123     captouch_configure();
124     adc_configure();
125     i2c_reset();
126
127
128     // Tasks Init
129     ring_init(&system_state.button_events);
130
131     // Start Scheduler
132     scheduler_configure();
133
134     // Enable Global Interrupts
135     EA = 1;
136
137     printf_tiny("Initialization Complete\n");
138
139     // Run tasks as they're dispatched
140     while(1)
141     {
142         if(task_flag_0)           // 1x sched_period
143         {
144             task_flag_0 = 0;
145             // Calculates step timing for the stepper
146             stepper_sequence_generator();
147         }
148         else if(task_flag_1)    // 2x sched_period
149         {
150             task_flag_1 = 0;
151             // Reads button states and puts them in the
152             // buffer

```

```

152                 task_buttons();
153             }
154         else if(task_flag_2) // 4x sched_period
155     {
156         task_flag_2 = 0;
157         // And timer
158         task_timer();
159         // Do ADC reading here
160         task_adc_samp();
161         // Do relay control here
162         task_therm_control();
163     }
164     else if(task_flag_3) // 8x sched_period
165     {
166         task_flag_3 = 0;
167         // Do main state machine here
168         task_ui();
169     }
170     else if(task_flag_4) // 16x sched_period
171     {
172         task_flag_4 = 0;
173         // Do LCD updates here
174         task_display();
175     }
176 }
177 }
178
179 /* ===== INIT INFO ===== */
180 uint8_t _sdcc_external_startup()
181 {
182     AUXR = AUXR_1024RAM_EXTRAM;
183     // Configure XRAM
184     // CKCON0 = PCAX2 | T2X2 | T1X2 | T0X2 | X2; // Ramming
185     // Speed
186     return 0;
187 }
188
189 /* INLINE FUNCTIONS FOR TASKS */
190
191 void task_buttons(void)
192 {
193     // The cap touch board will only report one button at a time
194     // So if the button changes, we know its been released
195     // If it stays the same, we consider it a long press
196
197     // Stores the last button seen
198     static button_t last_button = NO_KEY;
199     static uint8_t seen_count = 0;
200
201     // Get most recently pressed button

```

```

202     button_t button = captouch_poll_buttons();
203
204     // Button handling plan
205     // last      current      action
206     // NO_KEY    NO_KEY       nothing
207     // key       NO_KEY       output the last key
208     // NO_key    key         nothing
209     // key       key         if ==, increment the
210     //                 key count, if key count > thresh, output
211     //                                         if not equal
212     // output the last key
213
214     // Handle the buttons
215     if(button == NO_KEY)
216     {
217         // No active button
218         if(last_button != NO_KEY)
219         {
220             // If last button was active, then we just
221             // saw a falling edge
222             if(seen_count > LONG_TOUCH_MULTIPLIER)
223             {
224                 last_button += LONG_TOUCH_CODE;
225             }
226             ring_add_char(&system_state.button_events,
227                           last_button);
228         }
229         seen_count = 0; // if we have an active no-key we can
230         // 't be in a hold
231     }
232     else
233     {
234         // We have some active button
235         if(last_button == button)
236         {
237             // current button is the same as the last one
238             // we sampled
239             seen_count++; // increment a counter to
240             // catch long touches
241             if(seen_count > LONG_TOUCH_MULTIPLIER + 1)
242             {
243                 seen_count--; // this keeps
244                 // seen_count from rolling over
245             }
246         }
247     }
248
249     last_button = button; // Store for next time
250 }
251
252 void task_timer(void) __naked
253 {

```

```

246     // If the clock is running, add .02 seconds
247     // MM:SS.SS time[0, 1, 2]
248     __asm
249         // Jump out if timer isn't enabled
250         mov      dptr,#(_system_state + 0x0009)
251         movx    a,@dptr
252         jz      $00027
253
254         // Add .02 seconds to the clock (2 here since looking
255         // at hundredths)
256         mov      dptr,#(_system_state + 0x0005)
257         movx    a,@dptr
258         add      a,#0x02
259         da      a
260         movx    @dptr,a
261
262         // Add carry to seconds
263         mov      dptr,#(_system_state + 0x0004)
264         movx    a,@dptr
265         addc   a,#0x00
266         da      a
267
268         // If equal to 60, gotta set back to zero and set
269         // carry
270         mov      ac,c  // cjne messes with carry so
271         // we need to save it
272         cjne   a,#0x60,$00028
273         clr      a
274         setb    ac
275         $00028:
276         movx    @dptr,a
277         mov      c,ac
278
279         // Add carry to minutes
280         mov      dptr,#(_system_state + 0x0003)
281         movx    a,@dptr
282         addc   a,#0x00
283         da      a
284         movx    @dptr,a
285
286     }
287
288 void task_adc_samp(void)
289 {
290     // Read the current temperature and do a bit of moving
291     // average to smooth
292     system_state.temp_current = thermistor_convert(adc_reading(
293         adc_mode_single, adc_channel_0));
294 }
```

```

293
294 void task_therm_control(void)
295 {
296     typedef enum {temp_goal_set, temp_pulse, temp_coast}
297         temp_state_t;
298     static temp_state_t temp_state = temp_goal_set;
299     static uint8_t temp_coast_count;
300     static uint8_t local_goal;
301     static uint8_t last_temp;
302
303     if(system_state.temp_enable)
304     {
305         switch(temp_state)
306         {
307             case temp_goal_set:
308                 // Calc setpoint
309                 if(system_state.temp_current <
310                     system_state.temp_goal)
311                 {
312                     local_goal = system_state.
313                         temp_current + (
314                             system_state.temp_goal -
315                             system_state.temp_current
316                             )/4;
317                     temp_state = temp_pulse;
318                 }
319                 break;
320
321             case temp_pulse:
322                 // Turn on till you reach that point,
323                 // then turn off
324                 if(system_state.temp_current <
325                     local_goal)
326                 {
327                     RELAY_OUT = RELAY_ON;
328                 }
329                 else
330                 {
331                     RELAY_OUT = RELAY_OFF;
332                     temp_state = temp_coast;
333                 }
334                 break;
335
336             case temp_coast:
337                 // Coast till level out
338                 if(system_state.temp_current ==
339                     last_temp)
340                 {
341                     temp_coast_count++;
342                 }
343                 else
344                 {

```

```

336                     temp_coast_count = 0;
337                     last_temp = system_state.
338                         temp_current;
339
340                     if(temp_coast_count > 10)
341 {
342                         temp_coast_count = 0;
343                         temp_state = temp_goal_set;
344 }
345                     break;
346
347     }
348     else
349     {
350         RELAY_OUT = RELAY_OFF;
351     }
352 }
353
354 void task_display(void)
355 {
356     static ui_state_t lcd_last_state = home;
357
358     if(system_state.state == sleep)
359     {
360         lcd_clear();
361         lcd_last_state = sleep;
362     }
363     else
364     {
365         switch (system_state.state)
366     {
367         case egg:
368             if(lcd_last_state != egg)
369             {
370                 lcd_putstring(LCD_LINE_3, "
371                     Status: Settings");
372                 lcd_last_state = egg;
373             }
374             if(system_state.preset_updated)
375             {
376                 lcd_putstring(LCD_LINE_0, "
377                     ECG: ");
378                 lcd_putstring(LCD_LINE_0 + 6,
379                     system_state.preset_name
380                     );
381             }
382             break;
383
384         case tea:
385             if(lcd_last_state != tea)
386             {

```

```
383                         lcd_putstring(LCD_LINE_3, "
384                                         Status: Settings");
385                                         lcd_last_state = tea;
386 }
387 if(system_state.preset_updated)
388 {
389     lcd_putstring(LCD_LINE_0, "
390         TEA:           ");
391     lcd_putstring(LCD_LINE_0 + 6,
392                     system_state.preset_name
393                     );
394 }
395 break;

396 case water:
397     if(lcd_last_state != water)
398     {
399         lcd_putstring(LCD_LINE_3, "
400             Status: Settings");
401         lcd_last_state = water;
402     }
403     if(system_state.preset_updated)
404     {
405         lcd_putstring(LCD_LINE_0, "
406             WATER:          ");
407         lcd_putstring(LCD_LINE_0 + 6,
408                     system_state.preset_name
409                     );
410     }
411 break;

412 case run_heatup:
413     lcd_putstring(LCD_LINE_3, "Status:
414         Heating ");
415     lcd_last_state = run_heatup;
416     break;

417 case run_time:
418     lcd_putstring(LCD_LINE_3, "Status:
419         Cooking ");
420     lcd_last_state = run_heatup;
421     break;

422 case run_water:
423     lcd_putstring(LCD_LINE_3, "Status:
424         Heating");
425     lcd_last_state = run_water;
426     break;

427 default:
428     break;
429 }
```

```
424     static char lcd_data[4][8] = { "Tc:",  
425                                     "  
426                                         "  
427                                         "  
428                                         "  
429                                         "  
430                                         "  
431                                         "  
432                                         "  
433                                         "  
434                                         "  
435                                         "  
436                                         "  
437                                         "  
438                                         "  
439                                         "  
440                                         "  
441                                         "  
442                                         "  
443                                         "  
444                                         "  
445                                         "  
446                                         "  
447                                         "  
448                                         "  
449                                         "  
450                                         "  
451                                         "
```

```

452             lcd_data[2][6] = (system_state.time_current
453                 [1] & 0x0F) + '0';
454         }
455
456         // Update Current Temp
457         micro_sprintf(&lcd_data[0][3], system_state.
458             temp_current);
459         lcd_putstring(LCD_LINE_1, lcd_data[0]);
460     }
461
462     void task_ui(void)
463     {
464         static uint8_t inactive_count = 0;
465
466         // Grab the latest pressed key (if there is none, present
467         // NO_KEY)
468         button_t button;
469         if(ring_remove_char(&system_state.button_events, &button) !=
470             ERR_SUCCESS)
471         {
472             button = NO_KEY;
473
474             if(    (system_state.state == egg) ||
475                 (system_state.state == tea) ||
476                 (system_state.state == water))
477             {
478                 inactive_count++;
479                 if(inactive_count == 255)
480                 {
481                     system_state.state = sleep;
482                 }
483             }
484         }
485         else
486         {
487             inactive_count = 0;
488         }
489
490         // Switch through system state machine
491         switch(system_state.state)
492         {
493             case sleep:
494                 stepper_sleep();
495                 if(button != NO_KEY){system_state.state =
496                     home;}
497                 break;
498             case home:
499                 system_state.time_enable = false;
500                 system_state.temp_enable = false;

```

```
499         stepper_home();
500         system_state.state = egg;
501         select_preset(SQ_UL, &egg_preset);
502         break;
503
504     case egg:
505         if(button == PLAY)
506         {
507             system_state.restore_state = egg;
508             system_state.state = run_heatup;
509         }
510         else if (button == MODE)
511         {
512             system_state.state = tea;
513             select_preset(SQ_UL, &tea_preset);
514         }
515         else
516         {
517             select_preset(button, &egg_preset);
518         }
519         break;
520
521     case tea:
522         if(button == PLAY)
523         {
524             system_state.restore_state = tea;
525             system_state.state = run_heatup;
526         }
527         else if (button == MODE)
528         {
529             system_state.state = water;
530             select_preset(SQ_UL, &water_preset);
531         }
532         else
533         {
534             select_preset(button, &tea_preset);
535         }
536         break;
537
538     case water:
539         if(button == PLAY)
540         {
541             system_state.restore_state = water;
542             system_state.state = run_water;
543         }
544         else if (button == MODE)
545         {
546             system_state.state = egg;
547             select_preset(SQ_UL, &egg_preset);
548         }
549         else
550         {
```

```

551                     select_preset(button, &water_preset);
552                 }
553             break;
554
555         case run_heatup:
556             if(button == STOP)
557             {
558                 system_state.temp_enable = false;
559                 system_state.state = system_state.
560                     restore_state;
561             }
562
563             system_state.temp_enable = true;
564
565             if(      (system_state.temp_current >=
566                     system_state.temp_goal - 1) &&
567                     (system_state.temp_current <=
568                     system_state.temp_goal + 3))
569             {
570                 system_state.state = run_dropcar;
571             }
572             break;
573
574         case run_dropcar:
575             stepper_segment_add(-110, 20);
576             stepper_segment_add(-5, 0);
577             stepper_motion_enabled = true;
578             system_state.state = run_time;
579             break;
580
581         case run_time:
582             if(button == STOP)
583             {
584                 system_state.state = run_raisecar;
585             }
586
587             system_state.time_enable = true;
588
589             if(      (system_state.time_current[0] >=
590                     system_state.time_goal[0]) &&
591                     (system_state.time_current[1] >=
592                     system_state.time_goal[1]))
593             {
594                 system_state.state = run_raisecar;
595             }
596             break;
597
598         case run_raisecar:
599             stepper_segment_add(110, 20);
600             stepper_segment_add(5, 0);

```

```

598         stepper_motion_enabled = true;
599         system_state.time_enable = false;
600         system_state.temp_enable = false;
601         system_state.time_current[0] = 0;
602         system_state.time_current[1] = 0;
603         system_state.time_current[2] = 0;
604         system_state.state = system_state.
605             restore_state;
606             break;
607
608         case run_water:
609             if(button == STOP)
610             {
611                 system_state.state = system_state.
612                     restore_state;
613                     system_state.temp_enable = false;
614                     return;
615             }
616
617             system_state.temp_enable = true;
618
619             if(      (system_state.temp_current >=
620                     system_state.temp_goal - 1) &&
621                     (system_state.temp_current <=
622                     system_state.temp_goal + 3))
623             {
624                 system_state.state = system_state.
625                     restore_state;
626                     system_state.temp_enable = false;
627                     break;
628             }
629         }
630
631     void select_preset(button_t button, mode_presets_t* mode_preset)
632     {
633         preset_t* preset;
634         switch (button)
635         {
636             case SQ_UL:
637                 preset = &(mode_preset->SQ_UL);
638                 break;
639
640             case SQ_UR:
641                 preset = &(mode_preset->SQ_UR);
642                 break;
643
644             case SQ_LL:

```

```

645                     preset = &(mode_preset->SQ_LL);
646                     break;
647
648             case SQ_LR:
649                     preset = &(mode_preset->SQ_LR);
650                     break;
651
652             default:
653                     preset = NULL;
654                     break;
655     }
656
657     if(preset != NULL)
658     {
659         system_state.temp_goal = preset->temp;
660         system_state.time_goal = preset->time;
661         system_state.preset_name = preset->name;
662         system_state.preset_updated = true;
663     }
664 }
665
666 void presets_initialize(void)
667 {
668     //// EGGS
669     memcpy(egg_preset.SQ_UL.name, "Very Hard", sizeof("Very Hard"));
670     egg_preset.SQ_UL.temp = 210;
671     egg_preset.SQ_UL.time[0] = 0x09;
672     egg_preset.SQ_UL.time[1] = 0x00;
673
674     memcpy(egg_preset.SQ_UR.name, "Hard", sizeof("Hard"));
675     egg_preset.SQ_UR.temp = 200;
676     egg_preset.SQ_UR.time[0] = 0x08;
677     egg_preset.SQ_UR.time[1] = 0x00;
678
679     memcpy(egg_preset.SQ_LL.name, "Medium", sizeof("Medium"));
680     egg_preset.SQ_LL.temp = 180;
681     egg_preset.SQ_LL.time[0] = 0x10;
682     egg_preset.SQ_LL.time[1] = 0x00;
683
684     memcpy(egg_preset.SQ_LR.name, "Soft", sizeof("Soft"));
685     egg_preset.SQ_LR.temp = 167;
686     egg_preset.SQ_LR.time[0] = 0x12;
687     egg_preset.SQ_LR.time[1] = 0x00;
688
689     //// TEA
690     memcpy(tea_preset.SQ_UL.name, "Black", sizeof("Black"));
691     tea_preset.SQ_UL.temp = 206;
692     tea_preset.SQ_UL.time[0] = 0x05;
693     tea_preset.SQ_UL.time[1] = 0x00;
694
695

```

```

696     memcpy(tea_preset.SQ.UR.name, "Oolong", sizeof("Oolong"));
697     tea_preset.SQ.UR.temp = 203;
698     tea_preset.SQ.UR.time[0] = 0x04;
699     tea_preset.SQ.UR.time[1] = 0x00;
700
701     memcpy(tea_preset.SQ.LL.name, "White", sizeof("White"));
702     tea_preset.SQ.LL.temp = 185;
703     tea_preset.SQ.LL.time[0] = 0x03;
704     tea_preset.SQ.LL.time[1] = 0x00;
705
706     memcpy(tea_preset.SQ.LR.name, "Herbal", sizeof("Herbal"));
707     tea_preset.SQ.LR.temp = 203;
708     tea_preset.SQ.LR.time[0] = 0x06;
709     tea_preset.SQ.LR.time[1] = 0x00;
710
711     //// WATER
712     memcpy(water_preset.SQ.UL.name, "Scalding", sizeof("Scalding")
713         );
714     water_preset.SQ.UL.temp = 210;
715     water_preset.SQ.UL.time[0] = 0x00;
716     water_preset.SQ.UL.time[1] = 0x00;
717
718     memcpy(water_preset.SQ.UR.name, "Less Hot", sizeof("Less Hot")
719         );
720     water_preset.SQ.UR.temp = 190;
721     water_preset.SQ.UR.time[0] = 0x00;
722     water_preset.SQ.UR.time[1] = 0x00;
723
724     memcpy(water_preset.SQ.LL.name, "Warm", sizeof("Warm"));
725     water_preset.SQ.LL.temp = 150;
726     water_preset.SQ.LL.time[0] = 0x00;
727     water_preset.SQ.LL.time[1] = 0x00;
728
729     memcpy(water_preset.SQ.LR.name, "Tepid", sizeof("Tepid"));
730     water_preset.SQ.LR.temp = 120;
731     water_preset.SQ.LR.time[0] = 0x00;
732     water_preset.SQ.LR.time[1] = 0x00;
733
734     //// Clear the time
735     system_state.time_current[0] = 0;
736     system_state.time_current[1] = 0;
737
738     //// Flag the Display to Update
739     system_state.preset_updated = true;
740 }
741 void micro_sprintf(_far char* destination, uint8_t number) _naked
742 {
743     // // Only prints 8 bit numbers to a buffer
744     // uint8_t hund = number/100;
745     // number -= hund*100;

```

```

746     // uint8_t tens = number/10;
747     // number -= tens*10;
748
749     // destination[0] = hund + '0';
750     // destination[1] = tens + '0';
751     // destination[2] = number + '0';
752
753     // Suppress compiler warning about unused variables
754     destination;
755     number;
756
757 // dptr holds destination address
758 // number is at #_micro_sprintf_PARM_2 and in A
759     __asm
760     push    b
761
762     mov      b, #100          // Div by 100
763     div      ab             // A has the 100's
764     place, b has the remainder
765     add      a,#0x30         // Shift into ascii number
766     range
767     movx   @dptr, a          // Store it
768     inc      dptr           // Move address pointer
769     mov      a,b             // Move remainder to
770     A
771
772     mov      b, #10          // Div by 10
773     div      ab             // A has the 10's
774     place, b has the 1's
775     add      a,#0x30         // Shift into ascii number
776     range
777     movx   @dptr, a          // Store it
778     inc      dptr           // Move address pointer
779
780     mov      a,#'F'          // 'F' for degrees F
781     movx   @dptr, a          // Store it
782     inc      dptr           // Move address pointer
783
784     clr      a               // Null
785     movx   @dptr, a          // Store it
786
787     pop      b
788     ret
789     __endasm;
790 }

```

Listing C.3: Scheduler Header

```

1  /* T2 AS TASK SCHEDULER */
2  // Dominic Doty
3  // ECEN 5613
4
5
6  #ifndef SCHEDULER_H
7  #define SCHEDULER_H
8
9  /* INCLUDES */
10 #include <stdint.h>
11 #include <stdio.h>
12 #include "mcs51/at89c51ed2.h"
13 #include "pin_map.h"
14 #include "timer_driver.h"
15
16
17 /* DEFINES */
18 #define PERIPH_CLK_FREQ 11059200
19 #define CLK_PER_TICK    12
20 #define MS_PER_S          1000
21
22 /* TYPEDEFS */
23 // Div2 range is .002-142 ms
24 // Div6 range is .007-426 ms
25 typedef enum{
26     periph_clk_div6 = 0b00000000,
27     periph_clk_div2 = 0b00000010,
28     t_zero = 0b00000100,
29     ext_int = 0b00000110
30 }scheduler_base_freq_t;
31
32
33 /* GLOBALS */
34 extern volatile __bit task_flag_0;      // Flags for each service
35             period requested
36 extern volatile __bit task_flag_1;
37 extern volatile __bit task_flag_2;
38 extern volatile __bit task_flag_3;
39 extern volatile __bit task_flag_4;
40
41 /* PROTOTYPES */
42 // Set up Timer 2 as a scheduler, running at an interval of base
43             period (ms)
44 // tasks are dispatched in binary multiples
45 // t0 every base period, t1 every 2x base period, t2 every 4x base
46             period, etc.
47 void scheduler_configure(void);
48
49 // PCA ISR sets schedule flags when each period expires

```

```

48 void scheduler_ISR(void) __interrupt(TF2_VECTOR) __naked;
49
50 #endif // SCHEDULER_H

```

Listing C.4: Scheduler Source

```

1 /* T2 AS TASK SCHEDULER */
2 // Dominic Doty
3 // ECEN 5613
4
5
6 /* INCLUDES */
7 #include "scheduler.h"
8
9
10 /* DEFINES */
11 #define TICKS_PER_MS      9216
12
13
14 /* TYPEDEFS */
15
16
17 /* GLOBALS */
18 volatile __bit task_flag_0 = 0; // Flags for each service period
19 // requested
20 volatile __bit task_flag_1 = 0;
21 volatile __bit task_flag_2 = 0;
22 volatile __bit task_flag_3 = 0;
23 volatile __bit task_flag_4 = 0;
24
25 uint8_t int_count = 0;
26
27 /* FUNCTION DEFINITIONS */
28 void scheduler_configure(void)
29 {
30     // AUTORELOAD
31     // Set for .005s period
32     T2CON = 0x80;
33     RCAP2H = 0xEE;
34     RCAP2L = 0x00;
35     TL2 = RCAP2L;
36     TH2 = RCAP2H;
37
38     // Int setup and run
39     ET2 = 1;           // Enable overflow interrupt
40     TR2 = 1;           // Run
41 }
42 void scheduler_ISR(void) __interrupt(TF2_VECTOR) __naked
43 {
44     __asm
45     // Stack Regs

```

```

46      push    acc
47      push    dpl
48      push    dph
49      push    ar7
50      push    psw
51      mov     psw,#0x00
52
53      // Clear Int Flag
54      clr     _TF2
55
56      // Get the ISR count and increment it
57      mov     dptr,#_int_count
58      movx   a,@dptr
59      mov     r7,a
60      inc     a
61      movx   @dptr,a
62
63      // XOR the count with the previous count
64      xrl     a,r7
65
66      // Set dispatch bits
67      setb _task_flag_0          // Task 0 gets set every ISR run
68      rrc A                     // Just waste bit 0 since it
         changes every time (for task 0)
69
70      rrc A                     // Rotate right into carry (
         b0 -> carry)
71      orl C,_task_flag_1        // Or the flags together (ensure you
         don't clear something already set)
72      mov _task_flag_1,C        // Set the flag
73
74      rrc A
75      orl C,_task_flag_2
76      mov _task_flag_2,C
77      rrc A
78      orl C,_task_flag_3
79      mov _task_flag_3,C
80      rrc A
81      orl C,_task_flag_4
82      mov _task_flag_4,C
83
84      // Restore Regs
85      pop    psw
86      pop    ar7
87      pop    dph
88      pop    dpl
89      pop    acc
90
91      __endasm;
92  }

```

Listing C.5: Stepper Motor Driver Header

```

1  /* STEPPER EXECUTION CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  #ifndef STEPPER_H
6  #define STEPPER_H
7
8  /* INCLUDES */
9  #include <stdint.h>
10 #include <stdio.h>
11 #include <stdbool.h>
12 #include "mcs51/at89c51ed2.h"
13 #include "pin_map.h"
14 #include "timer_driver.h"
15 #include "ring_buffer.h"
16 #include "endstop.h"
17
18 /* DEFINES */
19 // Debug enable
20 // #define DEBUG_PRINT
21 // #define DEBUG_ASM_CHAR
22 #include "debug.h"
23
24 // Motion settings
25 #define STEP_PER_MM 25
26 #define STEP_ACCEL_DIV 32           // Bigger means slower,
                                  // smaller means faster, use powers of 2 for optimization
27 #define STEP_HOME_V 150            // mm/s
28 #define STEP_HOME_BACKOFF 2        // mm
29 #define STEP_TICKS_PER_S 921600 // timer ticks per 1 second
30
31 /* TYPEDEFS */
32
33
34 /* GLOBALS */
35 extern __bit stepper_motion_enabled;
36
37 /* PROTOTYPES */
38 // Configure Timer 1 and ring buffers
39 void stepper_configure(void);
40
41 // Add motion segments to the stepper queue
42 // dx is delta x in mm, vf is the desired final speed in mm/s
43 // the planner immediately tries to speed up to vf.
44 // If the segment isn't long enough to hit vf with specified accel,
45 // top speed will be arbitrary
46 void stepper_segment_add(int8_t dx, uint8_t vf);
47
48 // Takes segments from above function and turns them into timer
      intervals for the ISR

```

```

49 void stepper_sequence_generator(void);
50
51 // Put stepper to sleep (stops annoying noise)
52 void stepper_sleep(void);
53
54 // Home - drive into the endstop fast, then backoff, then slow. Sets
      system zero
55 // This is the only way to exit stepper sleeping state - since
      position is unknown as soon as you sleep
56 // THIS IS BLOCKING
57 void stepper_home(void);
58
59 // ISR for executing steps
60 void stepper_isr(void) __interrupt(TF0_VECTOR);
61
62 #endif // STEPPER_H

```

Listing C.6: Stepper Motor Driver Source

```

1 /* STEPPER EXECUTION CODE */
2 // Dominic Doty
3 // ECEN 5613
4
5
6 /* INCLUDES */
7 #include "stepper.h"
8
9
10 /* DEFINES */
11 // These step times, when passed to the ISR, mean special things
12 #define STEP_CODE_STOP 0x00
13 #define STEP_CODE_UP    0x01
14 #define STEP_CODE_DOWN   0x02
15
16 // Stepper DIR pin convention
17 #define STEP_DIR_UP        0x01
18 #define STEP_DIR_DOWN      0x00
19
20 // Sleep convention
21 #define STEP_SLEEP_EN     0x00
22 #define STEP_SLEEP_DIS    0x01
23
24 // This velocity (ticks/step, inverse velocity) is considered not
      moving
25 #define STEP_MIN_VEL     0xFFFFC
26
27 // Size of each double buffer must be even #
28 #define BUFFER_SIZE 16
29
30
31 /* TYPEDEFS */
32 typedef struct

```

```

33 {
34     int8_t dx;      // delta mm
35     uint8_t vf;    // final velocity of segment in mm/s
36 }stepper_segment_t;
37
38
39 /* GLOBALS */
40 __near uint8_t buffer_0[BUFFER_SIZE];
41 __near uint8_t buffer_1[BUFFER_SIZE];
42 __near uint8_t* __near isr_index = buffer_0;
43 __near uint8_t* __near generator_index = buffer_0;
44
45 __bit use_buffer_0 = 0; // Set when ready for ISR to use, cleared
46             // when ISR is done
47 __bit use_buffer_1 = 0;
48
49 ring_32_t step_seg_x;
50 ring_32_t step_seg_v;
51
52 __bit stepper_motion_enabled = false;
53
54 /* STATIC FUNCTION DECLARATIONS */
55 err_t double_buffer_add(uint16_t number);
56
57 /* FUNCTION DEFINITIONS */
58 void stepper_configure(void)
59 {
60     ring_init(&step_seg_x);
61     ring_init(&step_seg_v);
62
63     STEP_STEP = 1;
64     STEP_DIR = 0;
65     STEP_SLP = STEP_SLEEP_EN;
66
67     timer_init(timer_0, sixteen_bit_mode, true, false, false, 1,
68                false);
69     TR0 = 0;
70 }
71
72 void stepper_segment_add(int8_t dx, uint8_t vf)
73 {
74     ring_add_char(&step_seg_x, dx);
75     ring_add_char(&step_seg_v, vf);
76 }
77
78
79 void stepper_sequence_generator(void)
80 {
81     if(!stepper_motion_enabled)
82     {

```

```

83         return;
84     }
85
86     debug_print("Seqgen started\n", 0);
87
88     // Holds the segment we're currently working on
89     static stepper_segment_t command = {0,0};
90
91     // Current stepper velocity (inverse, ticks/step)
92     static uint16_t current_v = STEP_CODE_STOP;
93
94     err_t error;
95     static __bit direction_set = false;      // Stores if we've
96             set the movement direction
97     static int16_t dx_steps = 0;            // Stores the delta X
98             move in steps instead of mm
99     static uint16_t vf_steps = 0;           // Stores the final V
100            of the segment in ticks/step
101
102    // Blocks while planning till all moves in the buffer are
103        exhausted and the move is done
104    while(1)
105    {
106        // Translates to atomic JBC
107        if(endstop_flag)
108        {
109            endstop_flag = 0;
110
111            // Reset - dump buffers, stop ISR, reset
112                sequence state
113            TR0 = 0;
114            ring_init(&step_seg_x);
115            ring_init(&step_seg_v);
116            direction_set = false;
117            stepper_motion_enabled = false;
118            dx_steps = 0;
119            stepper_home();
120            return;
121        }
122
123        // If we don't have a command to work on yet, grab
124            one
125        if(dx_steps == 0)
126        {
127            debug_print("Get cmd\n", 0);
128
129            ring_remove_char(&step_seg_x, &command.dx);
130            debug_print("dxmm=%d\n", command.dx);
131
132            error = ring_remove_char(&step_seg_v, &
133                command.vf);
134            debug_print("Vfmm=%d\n", command.vf);
135
136        }
137
138        // Move the stepper
139        if(dx_steps > 0)
140        {
141            if(dx_steps <= 100)
142            {
143                dx_steps -= 100;
144                command.dx += 100;
145            }
146            else
147            {
148                dx_steps -= 100;
149                command.dx += 100;
150                if(dx_steps > 0)
151                {
152                    dx_steps -= 100;
153                    command.dx += 100;
154                }
155            }
156        }
157
158        if(dx_steps < 0)
159        {
160            if(dx_steps >= -100)
161            {
162                dx_steps += 100;
163                command.dx -= 100;
164            }
165            else
166            {
167                dx_steps += 100;
168                command.dx -= 100;
169                if(dx_steps < 0)
170                {
171                    dx_steps += 100;
172                    command.dx -= 100;
173                }
174            }
175        }
176
177        if(dx_steps == 0)
178        {
179            if(vf_steps > 0)
180            {
181                vf_steps -= 1;
182                command.vf += 1;
183            }
184            else
185            {
186                vf_steps += 1;
187                command.vf -= 1;
188            }
189        }
190
191        if(vf_steps == 0)
192        {
193            if(dx_steps > 0)
194            {
195                dx_steps -= 1;
196                command.dx += 1;
197            }
198            else
199            {
200                dx_steps += 1;
201                command.dx -= 1;
202            }
203        }
204
205        if(dx_steps == 0)
206        {
207            if(vf_steps > 0)
208            {
209                vf_steps -= 1;
210                command.vf += 1;
211            }
212            else
213            {
214                vf_steps += 1;
215                command.vf -= 1;
216            }
217        }
218
219        if(dx_steps == 0 && vf_steps == 0)
220        {
221            if(endstop_flag)
222            {
223                break;
224            }
225            else
226            {
227                if(dx_steps > 0)
228                {
229                    dx_steps -= 1;
230                    command.dx += 1;
231                }
232                else
233                {
234                    dx_steps += 1;
235                    command.dx -= 1;
236                }
237            }
238        }
239
240        if(dx_steps < 0 && vf_steps < 0)
241        {
242            if(endstop_flag)
243            {
244                break;
245            }
246            else
247            {
248                if(dx_steps >= -100)
249                {
250                    dx_steps += 100;
251                    command.dx -= 100;
252                }
253                else
254                {
255                    dx_steps -= 100;
256                    command.dx += 100;
257                }
258            }
259        }
260
261        if(dx_steps < 0 && vf_steps > 0)
262        {
263            if(endstop_flag)
264            {
265                break;
266            }
267            else
268            {
269                if(dx_steps <= 100)
270                {
271                    dx_steps -= 100;
272                    command.dx += 100;
273                }
274                else
275                {
276                    dx_steps += 100;
277                    command.dx -= 100;
278                }
279            }
280        }
281
282        if(dx_steps > 0 && vf_steps < 0)
283        {
284            if(endstop_flag)
285            {
286                break;
287            }
288            else
289            {
290                if(dx_steps <= -100)
291                {
292                    dx_steps += 100;
293                    command.dx -= 100;
294                }
295                else
296                {
297                    dx_steps -= 100;
298                    command.dx += 100;
299                }
300            }
301        }
302
303        if(dx_steps > 0 && vf_steps > 0)
304        {
305            if(endstop_flag)
306            {
307                break;
308            }
309            else
310            {
311                if(dx_steps <= 100)
312                {
313                    dx_steps -= 100;
314                    command.dx += 100;
315                }
316                else
317                {
318                    dx_steps += 100;
319                    command.dx -= 100;
320                }
321            }
322        }
323
324        if(dx_steps == 0)
325        {
326            if(vf_steps > 0)
327            {
328                vf_steps -= 1;
329                command.vf += 1;
330            }
331            else
332            {
333                vf_steps += 1;
334                command.vf -= 1;
335            }
336        }
337
338        if(vf_steps == 0)
339        {
340            if(dx_steps > 0)
341            {
342                dx_steps -= 1;
343                command.dx += 1;
344            }
345            else
346            {
347                dx_steps += 1;
348                command.dx -= 1;
349            }
350        }
351
352        if(dx_steps == 0)
353        {
354            if(vf_steps > 0)
355            {
356                vf_steps -= 1;
357                command.vf += 1;
358            }
359            else
360            {
361                vf_steps += 1;
362                command.vf -= 1;
363            }
364        }
365
366        if(dx_steps == 0 && vf_steps == 0)
367        {
368            if(endstop_flag)
369            {
370                break;
371            }
372            else
373            {
374                if(dx_steps > 0)
375                {
376                    dx_steps -= 1;
377                    command.dx += 1;
378                }
379                else
380                {
381                    dx_steps += 1;
382                    command.dx -= 1;
383                }
384            }
385        }
386
387        if(dx_steps < 0 && vf_steps < 0)
388        {
389            if(endstop_flag)
390            {
391                break;
392            }
393            else
394            {
395                if(dx_steps >= -100)
396                {
397                    dx_steps += 100;
398                    command.dx -= 100;
399                }
400                else
401                {
402                    dx_steps -= 100;
403                    command.dx += 100;
404                }
405            }
406        }
407
408        if(dx_steps < 0 && vf_steps > 0)
409        {
410            if(endstop_flag)
411            {
412                break;
413            }
414            else
415            {
416                if(dx_steps <= 100)
417                {
418                    dx_steps -= 100;
419                    command.dx += 100;
420                }
421                else
422                {
423                    dx_steps += 100;
424                    command.dx -= 100;
425                }
426            }
427        }
428
429        if(dx_steps > 0 && vf_steps < 0)
430        {
431            if(endstop_flag)
432            {
433                break;
434            }
435            else
436            {
437                if(dx_steps <= -100)
438                {
439                    dx_steps += 100;
440                    command.dx -= 100;
441                }
442                else
443                {
444                    dx_steps -= 100;
445                    command.dx += 100;
446                }
447            }
448        }
449
450        if(dx_steps > 0 && vf_steps > 0)
451        {
452            if(endstop_flag)
453            {
454                break;
455            }
456            else
457            {
458                if(dx_steps <= 100)
459                {
460                    dx_steps -= 100;
461                    command.dx += 100;
462                }
463                else
464                {
465                    dx_steps += 100;
466                    command.dx -= 100;
467                }
468            }
469        }
470
471        if(dx_steps == 0)
472        {
473            if(vf_steps > 0)
474            {
475                vf_steps -= 1;
476                command.vf += 1;
477            }
478            else
479            {
480                vf_steps += 1;
481                command.vf -= 1;
482            }
483        }
484
485        if(vf_steps == 0)
486        {
487            if(dx_steps > 0)
488            {
489                dx_steps -= 1;
490                command.dx += 1;
491            }
492            else
493            {
494                dx_steps += 1;
495                command.dx -= 1;
496            }
497        }
498
499        if(dx_steps == 0)
500        {
501            if(vf_steps > 0)
502            {
503                vf_steps -= 1;
504                command.vf += 1;
505            }
506            else
507            {
508                vf_steps += 1;
509                command.vf -= 1;
510            }
511        }
512
513        if(dx_steps == 0 && vf_steps == 0)
514        {
515            if(endstop_flag)
516            {
517                break;
518            }
519            else
520            {
521                if(dx_steps > 0)
522                {
523                    dx_steps -= 1;
524                    command.dx += 1;
525                }
526                else
527                {
528                    dx_steps += 1;
529                    command.dx -= 1;
530                }
531            }
532        }
533
534        if(dx_steps < 0 && vf_steps < 0)
535        {
536            if(endstop_flag)
537            {
538                break;
539            }
540            else
541            {
542                if(dx_steps >= -100)
543                {
544                    dx_steps += 100;
545                    command.dx -= 100;
546                }
547                else
548                {
549                    dx_steps -= 100;
550                    command.dx += 100;
551                }
552            }
553        }
554
555        if(dx_steps < 0 && vf_steps > 0)
556        {
557            if(endstop_flag)
558            {
559                break;
560            }
561            else
562            {
563                if(dx_steps <= 100)
564                {
565                    dx_steps -= 100;
566                    command.dx += 100;
567                }
568                else
569                {
570                    dx_steps += 100;
571                    command.dx -= 100;
572                }
573            }
574        }
575
576        if(dx_steps > 0 && vf_steps < 0)
577        {
578            if(endstop_flag)
579            {
580                break;
581            }
582            else
583            {
584                if(dx_steps <= -100)
585                {
586                    dx_steps += 100;
587                    command.dx -= 100;
588                }
589                else
590                {
591                    dx_steps -= 100;
592                    command.dx += 100;
593                }
594            }
595        }
596
597        if(dx_steps > 0 && vf_steps > 0)
598        {
599            if(endstop_flag)
600            {
601                break;
602            }
603            else
604            {
605                if(dx_steps <= 100)
606                {
607                    dx_steps -= 100;
608                    command.dx += 100;
609                }
610                else
611                {
612                    dx_steps += 100;
613                    command.dx -= 100;
614                }
615            }
616        }
617
618        if(dx_steps == 0)
619        {
620            if(vf_steps > 0)
621            {
622                vf_steps -= 1;
623                command.vf += 1;
624            }
625            else
626            {
627                vf_steps += 1;
628                command.vf -= 1;
629            }
630        }
631
632        if(vf_steps == 0)
633        {
634            if(dx_steps > 0)
635            {
636                dx_steps -= 1;
637                command.dx += 1;
638            }
639            else
640            {
641                dx_steps += 1;
642                command.dx -= 1;
643            }
644        }
645
646        if(dx_steps == 0)
647        {
648            if(vf_steps > 0)
649            {
650                vf_steps -= 1;
651                command.vf += 1;
652            }
653            else
654            {
655                vf_steps += 1;
656                command.vf -= 1;
657            }
658        }
659
660        if(dx_steps == 0 && vf_steps == 0)
661        {
662            if(endstop_flag)
663            {
664                break;
665            }
666            else
667            {
668                if(dx_steps > 0)
669                {
670                    dx_steps -= 1;
671                    command.dx += 1;
672                }
673                else
674                {
675                    dx_steps += 1;
676                    command.dx -= 1;
677                }
678            }
679        }
680
681        if(dx_steps < 0 && vf_steps < 0)
682        {
683            if(endstop_flag)
684            {
685                break;
686            }
687            else
688            {
689                if(dx_steps >= -100)
690                {
691                    dx_steps += 100;
692                    command.dx -= 100;
693                }
694                else
695                {
696                    dx_steps -= 100;
697                    command.dx += 100;
698                }
699            }
700        }
701
702        if(dx_steps < 0 && vf_steps > 0)
703        {
704            if(endstop_flag)
705            {
706                break;
707            }
708            else
709            {
710                if(dx_steps <= 100)
711                {
712                    dx_steps -= 100;
713                    command.dx += 100;
714                }
715                else
716                {
717                    dx_steps += 100;
718                    command.dx -= 100;
719                }
720            }
721        }
722
723        if(dx_steps > 0 && vf_steps < 0)
724        {
725            if(endstop_flag)
726            {
727                break;
728            }
729            else
730            {
731                if(dx_steps <= -100)
732                {
733                    dx_steps += 100;
734                    command.dx -= 100;
735                }
736                else
737                {
738                    dx_steps -= 100;
739                    command.dx += 100;
740                }
741            }
742        }
743
744        if(dx_steps > 0 && vf_steps > 0)
745        {
746            if(endstop_flag)
747            {
748                break;
749            }
750            else
751            {
752                if(dx_steps <= 100)
753                {
754                    dx_steps -= 100;
755                    command.dx += 100;
756                }
757                else
758                {
759                    dx_steps += 100;
760                    command.dx -= 100;
761                }
762            }
763        }
764
765        if(dx_steps == 0)
766        {
767            if(vf_steps > 0)
768            {
769                vf_steps -= 1;
770                command.vf += 1;
771            }
772            else
773            {
774                vf_steps += 1;
775                command.vf -= 1;
776            }
777        }
778
779        if(vf_steps == 0)
780        {
781            if(dx_steps > 0)
782            {
783                dx_steps -= 1;
784                command.dx += 1;
785            }
786            else
787            {
788                dx_steps += 1;
789                command.dx -= 1;
790            }
791        }
792
793        if(dx_steps == 0)
794        {
795            if(vf_steps > 0)
796            {
797                vf_steps -= 1;
798                command.vf += 1;
799            }
800            else
801            {
802                vf_steps += 1;
803                command.vf -= 1;
804            }
805        }
806
807        if(dx_steps == 0 && vf_steps == 0)
808        {
809            if(endstop_flag)
810            {
811                break;
812            }
813            else
814            {
815                if(dx_steps > 0)
816                {
817                    dx_steps -= 1;
818                    command.dx += 1;
819                }
820                else
821                {
822                    dx_steps += 1;
823                    command.dx -= 1;
824                }
825            }
826        }
827
828        if(dx_steps < 0 && vf_steps < 0)
829        {
830            if(endstop_flag)
831            {
832                break;
833            }
834            else
835            {
836                if(dx_steps >= -100)
837                {
838                    dx_steps += 100;
839                    command.dx -= 100;
840                }
841                else
842                {
843                    dx_steps -= 100;
844                    command.dx += 100;
845                }
846            }
847        }
848
849        if(dx_steps < 0 && vf_steps > 0)
850        {
851            if(endstop_flag)
852            {
853                break;
854            }
855            else
856            {
857                if(dx_steps <= 100)
858                {
859                    dx_steps -= 100;
860                    command.dx += 100;
861                }
862                else
863                {
864                    dx_steps += 100;
865                    command.dx -= 100;
866                }
867            }
868        }
869
870        if(dx_steps > 0 && vf_steps < 0)
871        {
872            if(endstop_flag)
873            {
874                break;
875            }
876            else
877            {
878                if(dx_steps <= -100)
879                {
880                    dx_steps += 100;
881                    command.dx -= 100;
882                }
883                else
884                {
885                    dx_steps -= 100;
886                    command.dx += 100;
887                }
888            }
889        }
890
891        if(dx_steps > 0 && vf_steps > 0)
892        {
893            if(endstop_flag)
894            {
895                break;
896            }
897            else
898            {
899                if(dx_steps <= 100)
900                {
901                    dx_steps -= 100;
902                    command.dx += 100;
903                }
904                else
905                {
906                    dx_steps += 100;
907                    command.dx -= 100;
908                }
909            }
910        }
911
912        if(dx_steps == 0)
913        {
914            if(vf_steps > 0)
915            {
916                vf_steps -= 1;
917                command.vf += 1;
918            }
919            else
920            {
921                vf_steps += 1;
922                command.vf -= 1;
923            }
924        }
925
926        if(vf_steps == 0)
927        {
928            if(dx_steps > 0)
929            {
930                dx_steps -= 1;
931                command.dx += 1;
932            }
933            else
934            {
935                dx_steps += 1;
936                command.dx -= 1;
937            }
938        }
939
940        if(dx_steps == 0)
941        {
942            if(vf_steps > 0)
943            {
944                vf_steps -= 1;
945                command.vf += 1;
946            }
947            else
948            {
949                vf_steps += 1;
950                command.vf -= 1;
951            }
952        }
953
954        if(dx_steps == 0 && vf_steps == 0)
955        {
956            if(endstop_flag)
957            {
958                break;
959            }
960            else
961            {
962                if(dx_steps > 0)
963                {
964                    dx_steps -= 1;
965                    command.dx += 1;
966                }
967                else
968                {
969                    dx_steps += 1;
970                    command.dx -= 1;
971                }
972            }
973        }
974
975        if(dx_steps < 0 && vf_steps < 0)
976        {
977            if(endstop_flag)
978            {
979                break;
980            }
981            else
982            {
983                if(dx_steps >= -100)
984                {
985                    dx_steps += 100;
986                    command.dx -= 100;
987                }
988                else
989                {
990                    dx_steps -= 100;
991                    command.dx += 100;
992                }
993            }
994        }
995
996        if(dx_steps < 0 && vf_steps > 0)
997        {
998            if(endstop_flag)
999            {
1000                break;
1001            }
1002            else
1003            {
1004                if(dx_steps <= 100)
1005                {
1006                    dx_steps -= 100;
1007                    command.dx += 100;
1008                }
1009                else
1010                {
1011                    dx_steps += 100;
1012                    command.dx -= 100;
1013                }
1014            }
1015        }
1016
1017        if(dx_steps > 0 && vf_steps < 0)
1018        {
1019            if(endstop_flag)
1020            {
1021                break;
1022            }
1023            else
1024            {
1025                if(dx_steps <= -100)
1026                {
1027                    dx_steps += 100;
1028                    command.dx -= 100;
1029                }
1030                else
1031                {
1032                    dx_steps -= 100;
1033                    command.dx += 100;
1034                }
1035            }
1036        }
1037
1038        if(dx_steps > 0 && vf_steps > 0)
1039        {
1040            if(endstop_flag)
1041            {
1042                break;
1043            }
1044            else
1045            {
1046                if(dx_steps <= 100)
1047                {
1048                    dx_steps -= 100;
1049                    command.dx += 100;
1050                }
1051                else
1052                {
1053                    dx_steps += 100;
1054                    command.dx -= 100;
1055                }
1056            }
1057        }
1058
1059        if(dx_steps == 0)
1060        {
1061            if(vf_steps > 0)
1062            {
1063                vf_steps -= 1;
1064                command.vf += 1;
1065            }
1066            else
1067            {
1068                vf_steps += 1;
1069                command.vf -= 1;
1070            }
1071        }
1072
1073        if(vf_steps == 0)
1074        {
1075            if(dx_steps > 0)
1076            {
1077                dx_steps -= 1;
1078                command.dx += 1;
1079            }
1080            else
1081            {
1082                dx_steps += 1;
1083                command.dx -= 1;
1084            }
1085        }
1086
1087        if(dx_steps == 0)
1088        {
1089            if(vf_steps > 0)
1090            {
1091                vf_steps -= 1;
1092                command.vf += 1;
1093            }
1094            else
1095            {
1096                vf_steps += 1;
1097                command.vf -= 1;
1098            }
1099        }
1100
1101        if(dx_steps == 0 && vf_steps == 0)
1102        {
1103            if(endstop_flag)
1104            {
1105                break;
1106            }
1107            else
1108            {
1109                if(dx_steps > 0)
1110                {
1111                    dx_steps -= 1;
1112                    command.dx += 1;
1113                }
1114                else
1115                {
1116                    dx_steps += 1;
1117                    command.dx -= 1;
1118                }
1119            }
1120        }
1121
1122        if(dx_steps < 0 && vf_steps < 0)
1123        {
1124            if(endstop_flag)
1125            {
1126                break;
1127            }
1128            else
1129            {
1130                if(dx_steps >= -100)
1131                {
1132                    dx_steps += 100;
1133                    command.dx -= 100;
1134                }
1135                else
1136                {
1137                    dx_steps -= 100;
1138                    command.dx += 100;
1139                }
1140            }
1141        }
1142
1143        if(dx_steps < 0 && vf_steps > 0)
1144        {
1145            if(endstop_flag)
1146            {
1147                break;
1148            }
1149            else
1150            {
1151                if(dx_steps <= 100)
1152                {
1153                    dx_steps -= 100;
1154                    command.dx += 100;
1155                }
1156                else
1157                {
1158                    dx_steps += 100;
1159                    command.dx -= 100;
1160                }
1161            }
1162        }
1163
1164        if(dx_steps > 0 && vf_steps < 0)
1165        {
1166            if(endstop_flag)
1167            {
1168                break;
1169            }
1170            else
1171            {
1172                if(dx_steps <= -100)
1173                {
1174                    dx_steps += 100;
1175                    command.dx -= 100;
1176                }
1177                else
1178                {
1179                    dx_steps -= 100;
1180                    command.dx += 100;
1181                }
1182            }
1183        }
1184
1185        if(dx_steps > 0 && vf_steps > 0)
1186        {
1187            if(endstop_flag)
1188            {
1189                break;
1190            }
1191            else
1192            {
1193                if(dx_steps <= 100)
1194                {
1195                    dx_steps -= 100;
1196                    command.dx += 100;
1197                }
1198                else
1199                {
1200                    dx_steps += 100;
1201                    command.dx -= 100;
1202                }
1203            }
1204        }
1205
1206        if(dx_steps == 0)
1207        {
1208            if(vf_steps > 0)
1209            {
1210                vf_steps -= 1;
1211                command.vf += 1;
1212            }
1213            else
1214            {
1215                vf_steps += 1;
1216                command.vf -= 1;
1217            }
1218        }
1219
1220        if(vf_steps == 0)
1221        {
1222            if(dx_steps > 0)
1223            {
1224                dx_steps -= 1;
1225                command.dx += 1;
1226            }
1227            else
1228            {
1229                dx_steps += 1;
1230                command.dx -= 1;
1231            }
1232        }
1233
1234        if(dx_steps == 0)
1235        {
1236            if(vf_steps > 0)
1237            {
1238                vf_steps -= 1;
1239                command.vf += 1;
1240            }
1241            else
1242            {
1243                vf_steps += 1;
1244                command.vf -= 1;
1245            }
1246        }
1247
1248        if(dx_steps == 0 && vf_steps == 0)
1249        {
1250            if(endstop_flag)
1251            {
1252                break;
1253            }
1254            else
1255            {
1256                if(dx_steps > 0)
1257                {
1258                    dx_steps -= 1;
1259                    command.dx += 1;
1260                }
1261                else
1262                {
1263                    dx_steps += 1;
1264                    command.dx -= 1;
1265                }
1266            }
1267        }
1268
1269        if(dx_steps < 0 && vf_steps < 0)
1270        {
1271            if(endstop_flag)
1272            {
1273                break;
1274            }
1275            else
1276            {
1277                if(dx_steps >= -100)
1278                {
1279                    dx_steps += 100;
1280                    command.dx -= 100;
1281                }
1282                else
1283                {
1284                    dx_steps -= 100;
1285                    command.dx += 100;
1286                }
1287            }
1288        }
1289
1290        if(dx_steps < 0 && vf_steps > 0)
1291        {
1292            if(endstop_flag)
1293            {
1294                break;
1295            }
1296            else
1297            {
1298                if(dx_steps <= 100)
1299                {
1300                    dx_steps -= 100;
1301                    command.dx += 100;
1302                }
1303                else
1304                {
1305                    dx_steps += 100;
1306                    command.dx -= 100;
1307                }
1308            }
1309        }
1310
1311        if(dx_steps > 0 && vf_steps < 0)
1312        {
1313            if(endstop_flag)
1314            {
1315                break;
1316            }
1317            else
1318            {
1319                if(dx_steps <= -100)
1320                {
1321                    dx_steps += 100;
1322                    command.dx -= 100;
1323                }
1324                else
1325                {
1326                    dx_steps -= 100;
1327                    command.dx += 100;
1328                }
1329            }
1330        }
1331
1332        if(dx_steps > 0 && vf_steps > 0)
1333        {
1334            if(endstop_flag)
1335            {
1336                break;
1337            }
1338            else
1339            {
1340                if(dx_steps <= 100)
1341                {
1342                    dx_steps -= 100;
1343                    command.dx += 100;
1344                }
1345                else
1346                {
1347                    dx_steps += 100;
1348                    command.dx -= 100;
1349                }
1350            }
1351        }
1352
1353        if(dx_steps == 0)
1354        {
1355            if(vf_steps > 0)
1356            {
1357                vf_steps -= 1;
1358                command.vf += 1;
1359            }
1360            else
1361            {
1362                vf_steps += 1;
1363                command.vf -= 1;
1364            }
1365        }
1366
1367        if(vf_steps == 0)
1368        {
1369            if(dx_steps > 0)
1370            {
1371                dx_steps -= 1;
1372                command.dx += 1;
1373            }
1374            else
1375            {
1376                dx_steps += 1;
1377                command.dx -= 1;
1378            }
1379        }
1380
1381        if(dx_steps == 0)
1382        {
1383            if(vf_steps > 0)
1384            {
1385                vf_steps -= 1;
1386                command.vf += 1;
1387            }
1388            else
1389            {
1390                vf_steps += 1;
1391                command.vf -= 1;
1392            }
1393        }
1394
1395        if(dx_steps == 0 && vf_steps == 0)
1396        {
1397            if(endstop_flag)
1398            {
1399                break;
1400            }
1401            else
1402            {
1403                if(dx_steps > 0)
1404                {
1405                    dx_steps -= 1;
1406                    command.dx += 1;
1407                }
1408                else
1409                {
1410                    dx_steps += 1;
1411                    command.dx -= 1;
1412                }
1413            }
1414        }
1415
1416        if(dx_steps < 0 && vf_steps < 0)
1417        {
1418            if(endstop_flag)
1419            {
1420                break;
1421            }
1422            else
1423            {
1424                if(dx_steps >= -100)
1425                {
1426                    dx_steps += 100;
1427                    command.dx -= 100;
1428                }
1429                else
1430                {
1431                    dx_steps -= 100;
1432                    command.dx += 100;
1433                }
1434            }
1435        }
1436
1437        if(dx_steps < 0 && vf_steps > 0)
1438        {
1439            if(endstop_flag)
1440            {
1441                break;
1442            }
1443            else
1444            {
1445                if(dx_steps <= 100)
1446                {
1447                    dx_steps -= 100;
1448                    command.dx += 100;
1449                }
1450                else
1451                {
1
```

```

128
129     if(error != ERR_SUCCESS)
130     {
131         // No more segments, quit
132         debug_print("No seg\n", 0);
133         stepper_motion_enabled = false;
134         direction_set = false;
135         current_v = 0;
136         return;
137     }
138
139     dx_steps = command.dx * STEP_PER_MM;
140     if(command.vf == 0)
141     {
142         vf_steps = STEP_MIN_VEL;
143     }
144     else
145     {
146         vf_steps = (STEP_TICKS_PER_S/
147                     STEP_PER_MM)/command.vf;
148     }
149     debug_print("dx=%d vf=%u\n", dx_steps,
150                 vf_steps);
151
152     // If we weren't able to set the direction yet
153     if(direction_set == false)
154     {
155         debug_print("Dir:\n", 0);
156         // Set the direction of movement for the new
157         // command
158         if(dx_steps > 0)
159         {
160             // Moving "forwards"
161             // Put the "forwards" code into the
162             // step time buffer
163             error = double_buffer_add(
164                         STEP_CODE_UP);
165             if(error != ERR_SUCCESS)
166             {
167                 TR0 = 1;
168                 continue; // Stop if
169                 // the buffer is full
170             }
171             direction_set = true;
172             debug_print("Dir FW\n", 0);
173         }
174         else
175         {
176             // Moving "backwards"

```

```

173 // Put the "backwards" code into the
174 // step time buffer
175 error = double_buffer_add(
176     STEP_CODE_DOWN);
177 if(error != ERR_SUCCESS)
178 {
179     TR0 = 1;
180     continue; // Stop if
181     the buffer is full
182 }
183 }

184

185 // If we're not moving, set velocity to min
186 if(current_v == STEP_CODE_STOP)
187 {
188     current_v = STEP_MIN_VEL;
189     debug_print("V=o, set slow\n", 0);
190 }

191

192 // Loop generating till full
193 while(dx_steps != 0)
194 {
195     error = double_buffer_add(~current_v); // Add to step buffer
196     if(error != ERR_SUCCESS)
197     {
198         debug_print("Bf full\n", 0);
199         TR0 = 1;
200         continue; // Stop if the buffer
201         is full
202     }
203
204     dx_steps > 0 ? dx_steps-- : dx_steps++; // decrement the number of remaining steps
205
206     if(current_v > vf_steps)
207     {
208         // Speed up
209         // Increase the velocity by the
210         // accelerator (inverse velocity,
211         // remember)
212         current_v -= current_v/STEP_ACCEL_DIV
213         ;
214
215         // If we overshot velocity, pin back
216         // to the specified V
217         if(current_v < vf_steps)
218         {
219             current_v = vf_steps;
220
221         }
222     }
223 }

```

```

215 }
216
217         debug_print("Spd+\n", 0);
218     }
219     else if(current_v < vf_steps)
220     {
221         // Slow down
222         uint16_t new_v = current_v + (
223             current_v/STEP_ACCEL_DIV);
224
225         // Decrease the velocity by the
226         // accelerator (inverse velocity,
227         // remember)
228         if(new_v < current_v)           ///
229             // Catch rollover
230         {
231             current_v = STEP_MIN_VEL;
232         }
233         else if (new_v > vf_steps)      ///
234             // Catch overshoot
235         {
236             current_v = vf_steps;
237         }
238         else
239             // Normal case
240         {
241             current_v += current_v/
242                 STEP_ACCEL_DIV;
243
244             debug_print("Spd-\n", 0);
245         }
246
247         debug_print("Stp gen fin\n", 0);
248
249         // Generated a whole segment, need to set direction
250         // for next one
251         direction_set = false;
252
253         // Start the timer for stepper execution, if it isn't
254         // already running
255         TR0 = 1;
256     }
257 }
```

```

258 void stepper_sleep(void)
259 {
260     STEP_SLP = STEP_SLEEP_EN;
261 }
262
263 void stepper_home(void)
264 {
265     // Set the endstop flag if the endstop is already triggered
266     if(ENDSTOP == 0)
267     {
268         endstop_flag = 1;
269     }
270
271     // Wake the stepper driver
272     STEP_SLP = STEP_SLEEP_DIS;
273
274     // Fast Home
275     debug_print("fast home enter\n",0);
276     STEP_DIR = STEP_DIR_UP;
277     while(1)
278     {
279         debug_print("stp\n",0);
280         // Do a step
281         STEP_STEP = 0;
282         __asm
283             NOP
284             NOP
285             NOP
286             NOP
287         __endasm;
288         STEP_STEP = 1;
289
290         // This translates to an atomic check/clear JBC
291         if(endstop_flag)
292         {
293             endstop_flag = 0;
294             break;
295         }
296
297         // Twiddle to create the right timing for homing
298         // speed
299         for(uint16_t i = 0; i<(1000000/(STEP_HOME_V *
300             STEP_PER_MM)); i++)
301         {
302             __asm
303                 NOP
304             __endasm;
305         }
306
307     // Backoff
308     debug_print("backoff enter\n",0);
309 }
```

```

308     STEP_DIR = STEP_DIR_DOWN;
309     while(ENDSTOP == 0)
310     {
311         // Do a step
312         debug_print("stp\n",0);
313         STEP_STEP = 0;
314         __asm
315             NOP
316             NOP
317             NOP
318             NOP
319             __endasm;
320         STEP_STEP = 1;
321
322         // Twiddle to create the right timing for homing
323         // speed
323         for(uint16_t i = 0; i<(1000000/(STEP_HOME_V *
324             STEP_PER_MM)); i++)
324         {
325             __asm
326                 NOP
327                 __endasm;
328         }
329     }
330 }
331
332 // Double Buffers
333 void stepper_isr(void) __interrupt(TF0_VECTOR) __naked
334 {
335     __asm
336         jb _endstop_flag,00065$
337         sjmp 00067$      // No endstop
338         00065$:          // Endstop
339         reti
340
341         // Stack used registers
342         00067$:
343         push acc
344         push ar0
345         push psw
346         mov psw,#0x00
347
348         // Stop the timer to prevent accidental rollovers
349         // while reloading
350         clr _TR0
351
352         // Load r0 with pointer
353         mov r0,_isr_index
354
355         // Check for direction change code
356         cjne @r0,#(STEP_CODE_UP>>8),00001$      //if the high
355         byte doesn't match, jump

```

```

356     inc r0
357     cjne @r0,#STEP_CODE_UP,00002$           //
358         check if Forwards flag
359         sjmp 00003$                         // match, jump to
360             Forwards
361             00002$:
362                 // no match
363                 cjne @r0,#STEP_CODE_DOWN,00004$      // check if
364                     Backwards flag
365                     sjmp 00005$                         // match, jump to
366                         Backwards
367                         00004$:
368                             // no match
369                             cjne @r0,#STEP_CODE_STOP,00019$    // check if Stop flag
370                             sjmp 00010$                         // match, jump to
371                                 Quit
372                                 00019$:
373                                     // no match
374                                     dec r0
375                                         // fix the buffer
376                                         pointer
377                                         sjmp 00001$                        

378                                         // Forwards code
379                                         00003$:
380                                         asm_put('f', 00099$)
381                                         setb _P1_4
382                                         inc r0
383                                         NOP
384                                         sjmp 00001$                        

385                                         // Backwards code
386                                         00005$:
387                                         asm_put('b', 00098$)
388                                         clr _P1_4
389                                         inc r0
390                                         NOP
391                                         sjmp 00001$                        

392                                         // Direction has been set
393                                         00001$:
394                                         asm_put('d', 00097$)

395                                         // Check if at end of buffer 0
396                                         cjne r0,#_buffer_0 + BUFFER_SIZE, 00015$      //
397                                         Not at end of B0, jump to check B1
398                                         sjmp 00006$                         // we're at the end of the buffer 0,
399                                         handle

```

```

392
393 // Check if at end of buffer 1
394 00015$:
395 cjne r0,#_buffer_1 + BUFFER_SIZE, 00008$      //
396           Not at end of B1 either, jump step&load
397 sjmp 00007$      // we're at the end of buffer 1,
398           handle
399
400 // Jumped here since r0 == end of buffer_0
401 00006$:
402     asm_put('o', 00096$)
403     clr _use_buffer_0      // let main know we're done
404           with 0
405     mov r0,#_buffer_1      // update buffer pointer
406     jb    _use_buffer_1,00008$    // jump to step and
407           reload if the buffer is ready
408 sjmp 00010$          // if it isn't, jump
409           to quit
410
411 // Jumped here since r0 == end of buffer_1
412 00007$:
413     asm_put('i', 00095$)
414     clr _use_buffer_1      // let main know we're done
415           with 0
416     mov r0,#_buffer_0      // update buffer pointer
417     jb    _use_buffer_0,00008$    // jump to step and
418           reload if the buffer is ready
419 sjmp 00010$          // if it isn't, jump
420           to quit
421
422 // Not at end of buffer
423 00008$:
424     asm_put('n', 00094$)
425
426 // Step
427     clr      _P1_3
428     NOP
429     NOP
430     NOP
431     NOP
432     setb    _P1_3
433
434 // Reload timer
435     mov      _TH0,@r0
436     inc      r0
437     mov      _TL0,@r0
438     inc      r0
439
440 // Restart the timer
441     setb    _TR0
442
443 // Store the pointer

```

```

436         asm_put('p', 00093$)
437         mov _isr_index,r0
438         sjmp 00011$
439
440         // Stop Moving and disable the timer
441         00010$:
442         asm_put('q', 00092$)
443         mov _isr_index,#_buffer_0           // Reset the pointer
444         clr _use_buffer_0                 // Reset the
445             buffer use flags
446         clr _use_buffer_1
447         clr TR0                         // Stop the timer
448
449         // Restore state
450         00011$:
451         asm_put('e', 00091$)
452         pop    psw
453         pop    ar0
454         pop    acc
455         reti
456     __endasm;
457 }
458
459
460 err_t double_buffer_add(uint16_t number)
461 {
462     // Check for space
463     if(generator_index == &buffer_0[BUFFER_SIZE])
464         // buffer_0 full
465     {
466         use_buffer_0 = 1;
467             // flag buffer_0 full
468         if(use_buffer_1 == 0)
469             // buffer_1 empty
470         {
471             debug_print("buffer 0 full , sw to 1\n",0);
472             generator_index = buffer_1;      // Next
473                 buffer is empty and this ones full,
474                 switch
475             }
476         else
477         {
478             debug_print("both full\n",0);
479             return ERR_FULL;
480         }
481     }
482     else if(generator_index == &buffer_1[BUFFER_SIZE])      //
483         buffer_1 full
484     {

```

```
479         use_buffer_1 = 1;
480             // flag buffer_1 full
480         if(use_buffer_0 == 0)
481             // buffer_0 empty
481         {
482             debug_print("buffer 1 full, sw to 0\n",0);
483             generator_index = buffer_0;      // Next
484             buffer is empty and this ones full,
484             switch
485             }
485         else
486         {
487             debug_print("both full\n",0);
488             return ERR_FULL;
489         }
490     }
491
492     // Add stuff to buffer since there's space
493     debug_print("added %u\n", number);
494     *generator_index = number>>8;
495     generator_index++;
496     *generator_index = number;
497     generator_index++;
498
499     return ERR_SUCCESS;
500 }
```

Listing C.7: Enstop Header

```

1  /* ENDSTOP ISR */
2  // Dominic Doty
3  // ECEN 5613
4
5
6  #ifndef ENDSTOP_H
7  #define ENDSTOP_H
8
9  /* INCLUDES */
10 #include <stdint.h>
11 #include <stdio.h>
12 #include "mcs51/at89c51ed2.h"
13 #include "pin_map.h"
14
15 /* GLOBALS */
16 extern volatile __bit endstop_flag;
17
18 /* PROTOTYPES */
19 // Setup INT0 for negative edge interrupt, set to input mode
20 void endstop_configure(void);
21
22 // Handle the endstop interrupt, set flag "endstop_flag"
23 void endstop_isr(void) __interrupt(IE0_VECTOR);
24
25 #endif // ENDSTOP_H

```

Listing C.8: Endstop Driver Source

```

1  /* ENDSTOP ISR */
2  // Dominic Doty
3  // ECEN 5613
4
5
6  /* INCLUDES */
7  #include "endstop.h"
8
9
10 /* GLOBALS */
11 volatile __bit endstop_flag = 0;
12
13 /* FUNCTION DEFINITIONS */
14 void endstop_configure(void)
15 {
16     ENDSTOP = 1;
17
18     IE0 = 0;           // Clear int?
19     IT0 = 1;           // Set to negative edge sensitive
20
21     PX0L;             // Set priority
22     IPH0 &= PX0H;

```

```
23     EX0 = 1;           // Enable int
24
25     // Set the endtop flag if the endstop is already triggered
26     if(ENDSTOP == 0)
27     {
28         endstop_flag = 1;
29     }
30 }
31
32
33 void endstop_isr(void) __interrupt(IE0_VECTOR)
34 {
35     endstop_flag = 1;
36     TR0 = 0;
37 }
```

Listing C.9: SPI Driver Header

```

1  /* SPI INTERFACE CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  #ifndef SPI_H
6  #define SPI_H
7
8  /* INCLUDES */
9  #include <stdint.h>
10 #include <stdio.h>
11 #include "mcs51/at89c51ed2.h"
12 #include "pin_map.h"
13
14 /* DEFINES */
15
16
17 /* TYPEDEFS */
18 typedef enum{
19         spi_ss_enable = 0,
20         spi_ss_disable = 0b00100000
21     }spi_ss_t;
22
23 typedef enum{
24         spi_mode_slave = 0,
25         spi_mode_master = 0b00010000
26     }spi_mode_t;
27
28 typedef enum{
29         spi_cpol_idle_high = 0b00001000,
30         spi_cpol_idle_low = 0
31     }spi_cpol_t;
32
33 typedef enum{
34         spi_cpha_sample_leave_idle = 0,
35         spi_cpha_sample_return_idle = 0b00000100
36     }spi_cpha_t;
37
38 typedef enum{
39         spi_baud_div2 = 0,
40         spi_baud_div4 = 1,
41         spi_baud_div8 = 2,
42         spi_baud_div16 = 3,
43         spi_baud_div32 = 0b10000000,    // Weird because
44             config bits are
45         spi_baud_div64 = 0b10000001,    // separated in the
46             SPCON register
47         spi_baud_div128 = 0b10000011
48     }spi_baud_div_t;
49
50 /* GLOBALS */

```

```

49
50
51 /* PROTOTYPES */
52 // Set up the SPI hardware
53 void spi_configure(
54     spi_ss_t ss_mode,
55     spi_mode_t mode,
56     spi_cpol_t cpol,
57     spi_cpha_t cpha,
58     spi_baud_div_t div);
59
60 // Perform a SPI transmission (and receive).
61 uint8_t spi_exchange(uint8_t tx);
62
63 #endif // SPI_H

```

Listing C.10: SPI Driver Source

```

1 /* SPI INTERFACE CODE */
2 // Dominic Doty
3 // ECEN 5613
4
5 /* INCLUDES */
6 #include "spi_driver.h"
7
8
9
10 /* FUNCTION DEFINITIONS */
11
12 void spi_configure(
13     spi_ss_t ss_mode,
14     spi_mode_t mode,
15     spi_cpol_t cpol,
16     spi_cpha_t cpha,
17     spi_baud_div_t div)
18 {
19     // disable spi (just in case)
20     SPCON &= ~SPEN;
21
22     // set gpio to high for MOSI MISO SCK
23     SPI_MOSI = 1;
24     SPI_MISO = 1;
25     SPI_SCK = 1;
26
27     // config SPCON baud, mode, cpol, cpha
28     SPCON = ss_mode | mode | cpol | cpha | div;
29
30     // enable
31     SPCON |= SPEN;
32 }
33
34 uint8_t spi_exchange(uint8_t tx)

```

```
35  {
36      SPDAT = tx;                                // Set the TX data
37      while(!(SPSTA & SPIF)); // Wait for TX/RX complete flag
38
39      // Idle for a bit to prevent super fast back to back
        transmissions
40      for(uint8_t i = 0; i < 150; i++)
41      {
42          __asm NOP __endasm;
43      }
44
45      return SPDAT;                            // Read RX data
46 }
```

Listing C.11: Capacitive Touch Driver Header

```

1  /* CAPACITIVE TOUCH INTERFACE CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  #ifndef CAPTOUCH_H
6  #define CAPTOUCH_H
7
8  /* INCLUDES */
9  #include <stdint.h>
10 #include <stdio.h>
11 #include "mcs51/at89c51ed2.h"
12 #include "pin_map.h"
13 #include "spi_driver.h"
14 #include "delay.h"
15
16 /* DEFINES */
17 #define LONG_TOUCH_CODE          0x80
18 #define LONG_TOUCH_MULTIPLIER    30
19
20 /* TYPEDEFS */
21 typedef enum{
22     PLAY = 6,
23     STOP = 0,
24     MODE = 5,
25     SQ_UL = 7,
26     SQ_UR = 10,
27     SQ_LL = 8,
28     SQ_LR = 9,
29     UP = 2,
30     DOWN = 3,
31     LEFT = 1,
32     RIGHT = 4,
33     PLAY_LONG = PLAY + LONG_TOUCH_CODE,
34     STOP_LONG = STOP + LONG_TOUCH_CODE,
35     MODE_LONG = MODE + LONG_TOUCH_CODE,
36     SQ_UL_LONG = SQ_UL + LONG_TOUCH_CODE,
37     SQ_UR_LONG = SQ_UR + LONG_TOUCH_CODE,
38     SQ_LL_LONG = SQ_LL + LONG_TOUCH_CODE,
39     SQ_LR_LONG = SQ_LR + LONG_TOUCH_CODE,
40     UP_LONG = UP + LONG_TOUCH_CODE,
41     DOWN_LONG = DOWN + LONG_TOUCH_CODE,
42     LEFT_LONG = LEFT + LONG_TOUCH_CODE,
43     RIGHT_LONG = RIGHT + LONG_TOUCH_CODE,
44     NO_KEY = 0xff
45 }button_t;
46
47 /* GLOBALS */
48
49
50 /* PROTOTYPES */

```

```

51 // Configure the captouch IC AT42QT1110, verify status is good
52 void captouch_configure(void);
53
54 // Get the status of all buttons. If there is a failure this will
55 // return 0
56 button_t captouch_poll_buttons(void);
57
58 #endif // CAPTOUCH_H

```

Listing C.12: Capacitive Touch Driver Source

```

1 /* CAPACITIVE TOUCH INTERFACE CODE */
2 // Dominic Doty
3 // ECEN 5613
4
5 /* INCLUDES */
6 #include "captouch_driver.h"
7
8
9 /* DEFINES */
10 #define CAPTOUCH_FIRST_KEY_REPT 0xC0
11 #define CAPTOUCH_ALL_KEYS_REPT 0xC1
12 #define CAPTOUCH_STATUS_REPT 0xC2
13 #define CAPTOUCH_QUICKSPI_ADDR 0x91
14 #define CAPTOUCH_QUICKSPI_DATA 0x04
15 #define CAPTOUCH_DIS_QUICKSPI 0x36
16 #define CAPTOUCH_CALIBRATE_ALL 0x03
17 #define CAPTOUCH_RESET 0x04
18 #define CAPTOUCH_IDLE 0x55
19 #define CAPTOUCH_STATUS_GOOD 0x80
20 #define CAPTOUCH_STATUS_MASK 0xF5
21 #define CAPTOUCH_MODE_SET 0x90
22 #define CAPTOUCH_MODE_GET 0xD0
23 #define CAPTOUCH_MODE 0xF0
24
25
26 /* TYPEDEFS */
27
28
29
30 /* FUNCTION DEFINITIONS */
31 void captouch_configure(void)
32 {
33     CS_CAPT = 1;    // Set CS in case it started low
34     CS_CAPT = 0;    // Assert CS
35
36     uint8_t rx;
37
38     // Check mode
39     spi_exchange(CAPTOUCH_MODE_GET);
40     rx = spi_exchange(0x00);

```

```

41     while(rx != CAPTOUCH_MODE)
42     {
43         rx = spi_exchange(CAPTOUCH_MODE_SET);
44         if(rx == CAPTOUCH_IDLE)
45         {
46             spi_exchange(CAPTOUCH_MODE);
47             spi_exchange(CAPTOUCH_MODE_GET);
48             rx = spi_exchange(0x00);
49         }
50     }
51     else
52     {
53         spi_exchange(0x00);
54     }
55 }
56
57     spi_exchange(CAPTOUCH_CALIBRATE_ALL);
58
59     CS_CAPT = 1; // Deassert CS
60 }
61
62 button_t captouch_poll_buttons(void)
63 {
64     uint8_t rx;
65     button_t button;
66
67     CS_CAPT = 0; // Assert CS
68
69     // Send nulls till the chip is idle (should be idle first tx)
70     do
71     {
72         rx = spi_exchange(0x00);
73     }while(rx != CAPTOUCH_IDLE);
74
75     spi_exchange(CAPTOUCH_FIRST_KEY_REPT); // command/status
76     rx = (spi_exchange(0x00));
77
78     CS_CAPT = 1; // Deassert CS
79
80     if(rx & 0x80)
81     {
82         // Key in detect
83         button = rx & 0x0F; // mask out the key number
84         only
85     }
86     else
87     {
88         button = NO_KEY;
89     }
90
91     return button;
92 }
```

Listing C.13: ADC Driver Header

```

1  /* ADC INTERFACE CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  #ifndef ADC_H
6  #define ADC_H
7
8  /* INCLUDES */
9  #include <stdint.h>
10 #include <stdio.h>
11 #include "mcs51/at89c51ed2.h"
12 #include "pin_map.h"
13 #include "spi_driver.h"
14
15 /* DEFINES */
16 #define ADC_8BIT           0x01 // set to 1 to only get MS 8
17                                bits, set to zero to get 10 bits
18
19 /* TYPEDEFS */
20
21 typedef enum{
22     adc_mode_single = 0x08,
23     adc_mode_differential = 0x00
24     }adc_mode_t;
25
26 // Also defines which channel is + input for differential mode
27 typedef enum{
28     adc_channel_0 = 0x00,
29     adc_channel_1 = 0x40
30     }adc_channel_t;
31
32 /* GLOBALS */
33
34 /* PROTOTYPES */
35 // Setup the ADC (clear the bus)
36 void adc_configure(void);
37
38 // Take an ADC reading in single or diff mode, select chan (or
39 // positive end)
40 // 8 bit or 10 bit mode is set by DEFINE in adc_driver
41 #if ADC_8BIT == 0x01
42     uint8_t adc_reading(adc_mode_t mode, adc_channel_t chan);
43 #else
44     uint16_t adc_reading(adc_mode_t mode, adc_channel_t chan);
45 #endif
46
47 #endif // ADC_H

```

Listing C.14: ADC Driver Source

```

1  /* ADC INTERFACE CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  /* INCLUDES */
6  #include "adc_driver.h"
7
8
9  /* DEFINES */
10 #define ADC_START_BIT    0x10
11 #define ADC_MSBF          0x02
12
13 /* FUNCTION DEFINITIONS */
14
15 void adc_configure(void)
16 {
17     //if device was powered up with cs low, it must be brought
18     //high and then low to reinint
19     CS_ADC = 1;
20     CS_ADC = 0;
21     CS_ADC = 1;
22 }
23
24 #if ADC_8BIT == 0x01
25     uint8_t adc_reading(adc_mode_t mode, adc_channel_t chan)
26 #else
27     uint16_t adc_reading(adc_mode_t mode, adc_channel_t chan)
28 #endif
29 {
30     // clocked out on falling edges and latched on rising edges
31     // clock idles low
32     // CPOL, CPHA = 0,0 or 1,1?
33
34     uint16_t result;
35
36     // CS goes low
37     CS_ADC = 0;
38
39     #if ADC_8BIT == 0x01
40         // 8 Bit Mode:
41         // TX: 0 0 0 SB MODE CHAN MSBF 0
42         // RX: DC
43         spi_exchange(ADC_START_BIT | mode | chan | ADC_MSBF);
44
45         // TX: 0x00
46         // RX: MS 8 bits
47         result = spi_exchange(0x00);
48
49     #else
50         uint8_t result_8;

```

```
50      // 10 Bit Mode:  
51      // TX: 0 SB MODE CHAN MSBF 0 0 0  
52      // RX: DC + MS 2 bits (in 0 and 1)  
53      result_8 = spi_exchange(ADC_START_BIT | mode | chan | msbf)  
54          <<2);  
55  
56      // TX: 0x00  
57      // RX: LS 8 bits  
58      result = spi_exchange(0x00);  
59  
60      // Combine MSB's with the LSB's  
61      result |= ((result_8 & 0x03) << 8);  
62  
63      #endif  
64  
65      // CS goes high  
66      CS_ADC = 1;  
67  
68      return result;  
69 }
```

Listing C.15: Thermistor Driver Header

```

1  /* THERMISTOR CONVERSION LOOKUP CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  #ifndef THERMISTOR_H
6  #define THERMISTOR_H
7
8  /* INCLUDES */
9  #include <stdint.h>
10 #include <stdio.h>
11 #include "mcs51/at89c51ed2.h"
12 #include "pin_map.h"
13
14 /* DEFINES */
15
16
17 /* TYPEDEFS */
18
19
20 /* GLOBALS */
21
22
23 /* PROTOTYPES */
24
25 // Takes a raw ADC reading and converts it to a temperature in
   Fahrenheit
26 uint8_t thermistor_convert(uint8_t adc_reading);
27
28
29 #endif // THERMISTOR_H

```

Listing C.16: Thermistor Driver Source

```

1  /* THERMISTOR CONVERSION LOOKUP CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  /* INCLUDES */
6  #include "thermistor.h"
7
8
9  /* FUNCTION DEFINITIONS */
10
11 uint8_t thermistor_convert(uint8_t adc_reading)
12 {
13     // Using simple linear conversion for now, maybe tune this
       later if not accurate
14     uint8_t temperature = 274 - ( ( ((uint16_t)adc_reading) * 31)
15                               /32);
16     return temperature;

```

16 }

Listing C.17: Debug Printing Header

```
1  /* DEBUG PRINTING CODE */
2  // Dominic Doty
3  // ECEN 5613
4
5  #ifndef DEBUG_H
6  #define DEBUG_H
7
8  #ifdef DEBUG_PRINT
9      #include "serial.h"
10
11     #define debug_print(fmt, ...) printf_tiny(fmt, __VA_ARGS__)
12 #else
13     #define debug_print(fmt, ...) // Nothing
14 #endif
15
16 #ifdef DEBUG_ASM_CHAR
17     #define HASH_LIT #
18     #define HASH() HASH_LIT
19
20     #define asm_put(CHAR, LABEL) \
21         LABEL: \
22             jnb    _TI,LABEL \
23             clr    _TI \
24             mov    _SBUF, HASH()CHAR
25 #else
26     #define asm_put(CHAR, LABEL) // Nothing
27 #endif
28
29 #endif // DEBUG_H
```

Listing C.18: Delay Functions Header

```

1  /* ===== BASIC DELAY ===== */
2 // Dominic Doty
3 // ECEN 5613
4
5 #ifndef DELAY_H
6 #define DELAY_H
7
8 /* INCLUDES */
9 #include <stdint.h>
10 #include <stdio.h>
11
12 /* DEFINES */
13
14 /* TYPEDEFS */
15
16 /* GLOBALS */
17
18 /* PROTOTYPES */
19 void seventyms_delay(void);
20
21 #endif // DELAY_H

```

Listing C.19: Delay Functions Source

```

1  /* ===== BASIC DELAY ===== */
2 // Dominic Doty
3 // ECEN 5613
4
5
6 /* ===== HEADER ===== */
7 #include "delay.h"
8
9
10
11 /* ===== FUNCTION DEFINITIONS ===== */
12 void seventyms_delay(void)
13 {
14     for(uint8_t i = 0; i < 255; i++)
15     {
16         for(uint8_t j = 0; j < 255; j++)
17         {
18             __asm
19                 NOP;
20             __endasm;
21         }
22     }
23 }

```

D

PART DATASHEETS

D.1 RELAY DATASHEET



General Purpose
Power PCB Relays

Power PCB Relay RZF

- 1 pole, 16A, 1 form A (NO)
- Coil power 530mW
- Reinforced insulation (EN 61810, 60335, 60730)
- Ambient temperature up to 85°C
- Quick connect terminals for load
- Low mounted height of 17.9mm (27.6mm with quick connects)
- WG version with material in accordance with IEC 60335-1

Typical applications
Microwave ovens, water heaters, ovens, industrial equipment.



Approvals

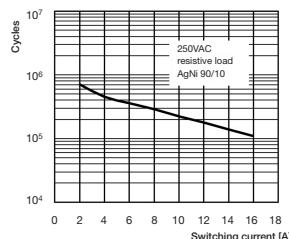
VDE 40046175, UL E214025, CQC 17002175064

Technical data of approved types on request.

Contact Data

Contact arrangement	1 form A (NO)
Rated voltage	250VAC
Max. switching voltage	400VAC
Rated current	16A
Limited making current, form A contact, max. 4 s, duty factor 10%	16A
Switching power	4000VA
Contact material	AgNi
Min. recommended contact load	100mA, 5VDC
Frequency of operation, with/without load	360/18000h ⁻¹
Operate/release time max.	8ms/6ms
Bounce time max.	4ms
Electrical endurance	
16A, 250VAC, resistive, 23°C	100x10 ³ ops.
16A, 250VAC, resistive, 85°C	50x10 ³ ops.
Contact ratings	16A, 250VAC, resistive, 23°C, 100x10 ³ ops.
	16A, 250VAC, resistive, 85°C, 100x10 ³ ops.
Mechanical endurance	10x10 ⁶ operations

Electrical endurance



Coil Data

Coil voltage range 5 to 48VDC

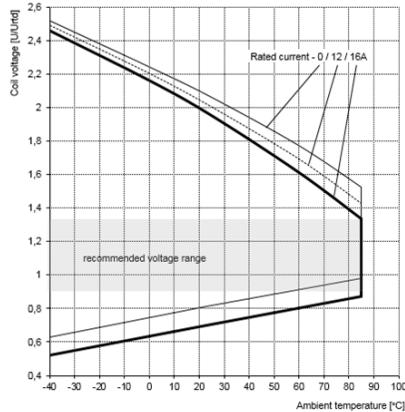
Coil operative range, IEC 61810 2

Coil insulation system according UL class F

Coil code	Rated voltage VDC	Operate voltage VDC	Release voltage VDC	Coil resistance $\Omega \pm 10\%$	Rated coil power mW
005	5	3.5	0.5	47.2	530
006	6	4.2	0.6	66.6	530
009	9	6.3	0.9	152.8	530
012	12	8.4	1.2	271.7	530
018	18	12.6	1.8	611	530
024	24	16.8	2.4	1086	530
048	48	33.6	4.8	4347	530

All figures are given for coil without pre-energization, at ambient temperature +23°C.

Coil Operating Range



01-2018, Revised
www.te.com
© 2018 TE Connectivity Ltd. family
of companies. All rights reserved.

Catalog and product specification according
to IEC 61810-1 and to be used only together
with the 'Definitions' section.

Catalog and product data is subject to the
terms of the disclaimer and all chapters of
the 'Definitions' section, available at
<http://relays.te.com/definitions>

Catalog product data, 'Definitions' section,
application notes and all specifications are
subject to change.



General Purpose Power PCB Relays

Power PCB Relay RZF (Continued)

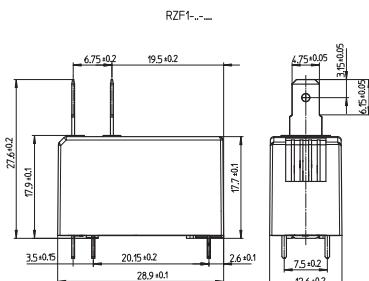
Insulation Data

Initial dielectric strength between open contacts between contact and coil	1000V _{rms} 5000V _{rms}
Initial surge withstand voltage between contact and coil	10000V
Clearance/creepage between contact and coil	≥ 5.5/8mm
Material group of insulation parts	III
Tracking index of relay base	PTI 300

Other Data

Material compliance: EU RoHS/ELV, China RoHS, REACH, Halogen content refer to the Product Compliance Support Center at www.te.com/customersupport/rhssupportcenter	
Resistance to heat and fire	According EN 60335-1, par. 30
Ambient temperature	-40 to 85°C
Category of environmental protection	IEC 61810
Vibration resistance (functional), 3 to 100Hz	>20g
Shock resistance (functional)	>10g
Shock resistance (destructive)	>100g
Terminal type	PCB-THT, quick connect for load side
Weight	11g
Resistance to soldering heat THT	270°C/10s
IEC 60068-2-20	tube/20 pcs.
Packaging/unit	box/500 pcs.

Dimensions

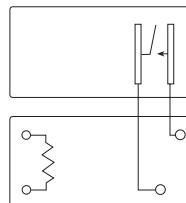


Terminal dimensions: Coil-Terminal: 0.5±0.025
Contact-Terminal: 0.5±0.01 x 0.8±0.05

- All terminal dimensions valid for the unlined terminal
- For the tin-plating of the pins add +0.1mm for the width, thickness or diameter.

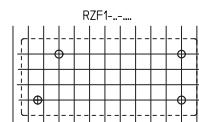
Terminal assignment

Bottom view on solder pins



PCB layout

Bottom view on solder pins



grid pattern: 250 to 254
hole diameter: Ø1.3 ±0.1
Bottom view on solder pins
dimensions in mm

2

01-2018 Revised
www.te.com
© 2018 TE Connectivity Ltd. family
of companies. All rights reserved.

Catalog and product specification according to IEC 61810-1 and to be used only together with the 'Definitions' section.

Catalog and product data is subject to the terms of the disclaimer and all chapters of the 'Definitions' section, available at <http://relays.te.com/definitions>

Catalog product data, 'Definitions' section, application notes and all specifications are subject to change.



General Purpose
Power PCB Relays

Power PCB Relay RZF (Continued)

Product code structure	Typical product code	RZF	1	-1A	4	-L	012	000
Type	RZF	Power PCB Relay RZF						
Version	1	16A, 3.5mm pinning, 4 PCB pins						
Contact configuration	1A	1 form A (NO) contact						
Contact material	4	AgNi 90/10	6	AgNi 90/10, special version				
Coil version	L	DC coil, 530mW						
Coil voltage		Coil code: please refer to coil versions table						
Suffix	000	Standard, RT II, Reinforced flux proof, IEC 60335-1 Compliant		Other	Special			

Note: May be followed by up to five additional characters for manufacturer internal identification.

Product code	Version	Contact	Cont.material	Coil power	Coil voltage	Sealing	Part number
RZF1-1A4-L005	4 PCB pins	1 form A (NO)	AgNi 90/10 (std.)	530mW	5VDC	Flux proof	1833011-1
RZF1-1A4-L006					6VDC		1833011-2
RZF1-1A4-L009					9VDC		1833011-3
RZF1-1A4-L012					12VDC		1833011-4
RZF1-1A4-L018					18VDC		1833011-5
RZF1-1A4-L024					24VDC		1833011-6
RZF1-1A4-L043					48VDC		1833011-7
RZF1-1A6-L005			AgNi 90/10 (spl.)		5VDC		2-1833011-8
RZF1-1A6-L006					6VDC		1-1833011-5
RZF1-1A6-L009					9VDC		1-1833011-6
RZF1-1A6-L012					12VDC		1-1833011-7
RZF1-1A6-L018					18VDC		1-1833011-8
RZF1-1A6-L024					24VDC		1-1833011-9
RZF1-1A6-L048					48VDC		2-1833011-0



General Purpose
Power PCB Relays

Power PCB Relay RZF (Continued)

te.com

TE Connectivity and TE connectivity (logo) are trademarks.
Other products, logos and company names mentioned herein may be trademarks of their respective owners.

The information given herein, including drawings, illustrations and schematics which are intended for illustration purposes only, is believed to be reliable. However, TE Connectivity makes no warranties as to its accuracy or completeness and disclaims any liability in connection with its use. TE Connectivity's obligations shall only be as set forth in TE Connectivity's Standard Terms and Conditions of Sale for this product and in no case will TE Connectivity be liable for any incidental, indirect or consequential damages arising out of the sale, resale, use or misuse of the product. Users of TE Connectivity products should make their own evaluation to determine the suitability of each such product for the specific application.

4

01-2018 Revised
www.te.com
© 2018 TE Connectivity Ltd. family
of companies. All rights reserved.

Catalog and product specification according
to IEC 61810-1 and to be used only together
with the 'Definitions' section.

Catalog and product data is subject to the
terms of the disclaimer and all chapters of
the 'Definitions' section, available at
<http://relays.te.com/definitions>

Catalog product data, 'Definitions' section,
application notes and all specifications are
subject to change.

D.2 ADC DATASHEET



MCP3002

2.7V Dual Channel 10-Bit A/D Converter with SPI Serial Interface

- 10-bit resolution
- ± 1 LSB maximum DNL
- ± 1 LSB maximum INL
- Analog inputs programmable as single-ended or pseudo-differential pairs
- On-chip sample and hold
- SPI serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- 200 kspS max sampling rate at $V_{DD} = 5V$
- 75 kspS max sampling rate at $V_{DD} = 2.7V$
- Low power CMOS technology:
 - 5 nA typical standby current, 2 μ A maximum
 - 550 μ A maximum active current at 5V
- Industrial temperature range: -40°C to +85°C
- 8-pin MSOP, PDIP, SOIC and TSSOP packages

The MCP3002 is a successive approximation 10-bit analog-to-digital (A/D) converter with on-board sample and hold circuitry.

The MCP3002 is programmable to provide a single pseudo-differential input pair or dual single-ended inputs. Differential Nonlinearity (DNL) and Integral Nonlinearity (INL) are both specified at ± 1 LSB. Communication with the device is done using a simple serial interface compatible with the SPI protocol. The device is capable of conversion rates of up to 200 kspS at 5V and 75 kspS at 2.7V.

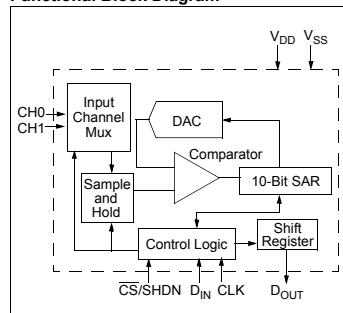
The MCP3002 operates over a broad voltage range, 2.7V to 5.5V. Low-current design permits operation with a typical standby current of 5 nA and a typical active current of 375 μ A.

The MCP3002 is offered in 8-pin MSOP, PDIP, TSSOP and 150 mil SOIC packages.

Applications

- Sensor Interface
- Process Control
- Data Acquisition
- Battery Operated Systems

Functional Block Diagram



Package Types

MSOP, PDIP, SOIC, TSSOP	
$\overline{CS}/SHDN$	1
CH0	2
CH1	3
V _{SS}	4
	8
	7
	6
	5
V _{DD/V_{REF}}	
CLK	
D _{OUT}	
D _{IN}	

MCP3002

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings †

V _{DD}	7.0V
All Inputs and Outputs w.r.t. V _{SS}	-0.6V to V _{DD} + 0.6V
Storage Temperature	-65°C to +150°C
Ambient temperature with power applied	-65°C to +150°C
ESD Protection On All Pins (HBM)	> 4 kV

† Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

All parameters apply at V_{DD} = 5V, T_A = -40°C to +85°C, f_{SAMPLE} = 200 ksps and f_{CLK} = 16*f_{SAMPLE}, unless otherwise noted.

Typical values apply for V_{DD} = 5V, T_A = +25°C, unless otherwise noted.

PARAMETER	SYM	MIN	TYP	MAX	UNITS	CONDITIONS
Conversion Rate:						
Conversion Time	T _{CONV}	—	—	10	clock cycles	
Analog Input Sample Time	T _{SAMPLE}		1.5		clock cycles	
Throughput Rate	f _{SAMPLE}	—	—	200 75	ksps ksps	V _{DD} = 5V V _{DD} = 2.7V
DC Accuracy:						
Resolution			10		bits	
Integral Nonlinearity	INL	—	±0.5	±1	LSB	
Differential Nonlinearity	DNL	—	±0.25	±1	LSB	No missing codes over temperature
Offset Error		—	—	±1.5	LSB	
Gain Error		—	—	±1	LSB	
Dynamic Performance:						
Total Harmonic Distortion	THD	—	-76	—	dB	V _{IN} = 0.1V to 4.9V@1 kHz
Signal to Noise and Distortion (SINAD)	SINAD	—	61	—	dB	V _{IN} = 0.1V to 4.9V@1 kHz
Spurious Free Dynamic Range	SFDR	—	78	—	dB	V _{IN} = 0.1V to 4.9V@1 kHz
Analog Inputs:						
Input Voltage Range for CH0 or CH1 in Single-ended Mode		V _{SS}	—	V _{DD}	V	
Input Voltage Range for IN+ in Pseudo-Differential Mode	IN+	IN-	—	V _{DD} +IN-		
Input Voltage Range for IN- in Pseudo-Differential Mode	IN-	V _{SS} -100	—	V _{SS} +100	mV	
Leakage Current		—	0.001	±1	µA	
Switch Resistance	R _{SS}	—	1K	—	Ω	See Figure 4-1
Sample Capacitor	C _{SAMPLE}	—	20	—	pF	See Figure 4-1

Note 1: This parameter is established by characterization and not 100% tested.

2: The sample cap will eventually lose charge, especially at elevated temperatures, therefore f_{CLK} ≥ 10 kHz for temperatures at or above 70°C.

MCP3002**ELECTRICAL CHARACTERISTICS (CONTINUED)**

All parameters apply at $V_{DD} = 5V$, $T_A = -40^\circ C$ to $+85^\circ C$, $f_{SAMPLE} = 200$ kspS and $f_{CLK} = 16 \cdot f_{SAMPLE}$, unless otherwise noted.

Typical values apply for $V_{DD} = 5V$, $T_A = +25^\circ C$, unless otherwise noted.

PARAMETER	SYM	MIN	TYP	MAX	UNITS	CONDITIONS
Digital Input/Output:						
Data Coding Format						
High Level Input Voltage	V_{IH}	0.7 V_{DD}	—	—	V	
Low Level Input Voltage	V_{IL}	—	—	0.3 V_{DD}	V	
High Level Output Voltage	V_{OH}	4.1	—	—	V	$I_{OH} = -1$ mA, $V_{DD} = 4.5V$
Low Level Output Voltage	V_{OL}	—	—	0.4	V	$I_{OL} = 1$ mA, $V_{DD} = 4.5V$
Input Leakage Current	I_{LI}	-10	—	10	μA	$V_{IN} = V_{SS}$ or V_{DD}
Output Leakage Current	I_{LO}	-10	—	10	μA	$V_{OUT} = V_{SS}$ or V_{DD}
Pin Capacitance (All Inputs/Outputs)	C_{IN}, C_{OUT}	—	—	10	pF	$V_{DD} = 5.0V$ (Note 1) $T_A = 25^\circ C$, $f = 1$ MHz
Timing Parameters:						
Clock Frequency	f_{CLK}	—	—	3.2 1.2	MHz MHz	$V_{DD} = 5V$ (Note 2) $V_{DD} = 2.7V$ (Note 2)
Clock High Time	t_{HII}	140	—	—	ns	
Clock Low Time	t_{LO}	140	—	—	ns	
CS Fall To First Rising CLK Edge	t_{SUCS}	100	—	—	ns	
Data Input Setup Time	t_{SU}	50	—	—	ns	
Data Input Hold Time	t_{HD}	50	—	—	ns	
CLK Fall To Output Data Valid	t_{DO}	—	—	125 200	ns ns	$V_{DD} = 5V$, see Figure 1-2 $V_{DD} = 2.7V$, see Figure 1-2
CLK Fall To Output Enable	t_{EN}	—	—	125 200	ns ns	$V_{DD} = 5V$, see Figure 1-2 $V_{DD} = 2.7V$, see Figure 1-2
CS Rise To Output Disable	t_{DIS}	—	—	100	ns	See Test Circuits, Figure 1-2 Note 1
CS Disable Time	t_{CSH}	310	—	—	ns	
D_{OUT} Rise Time	t_R	—	—	100	ns	See Test Circuits, Figure 1-2 Note 1
D_{OUT} Fall Time	t_F	—	—	100	ns	See Test Circuits, Figure 1-2 Note 1
Power Requirements:						
Operating Voltage	V_{DD}	2.7	—	5.5	V	
Operating Current	I_{DD}	—	525 300	650	μA	$V_{DD} = 5.0V$, D_{OUT} unloaded $V_{DD} = 2.7V$, D_{OUT} unloaded
Standby Current	I_{DDS}	—	0.005	2	μA	$CS = V_{DD} = 5.0V$

Note 1: This parameter is established by characterization and not 100% tested.

2: The sample cap will eventually lose charge, especially at elevated temperatures, therefore $f_{CLK} \geq 10$ kHz for temperatures at or above $70^\circ C$.

MCP3002

TEMPERATURE CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, $V_{DD} = +2.7V$ to $+5.5V$, $V_{SS} = GND$.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Temperature Ranges						
Specified Temperature Range	T_A	-40	—	+85	°C	
Operating Temperature Range	T_A	-40	—	+85	°C	
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistances						
Thermal Resistance, 8L-MSOP	θ_{JA}	—	211	—	°C/W	
Thermal Resistance, 8L-PDIP	θ_{JA}	—	89.5	—	°C/W	
Thermal Resistance, 8L-SOIC	θ_{JA}	—	149.5	—	°C/W	
Thermal Resistance, 8L-TSSOP	θ_{JA}	—	139	—	°C/W	

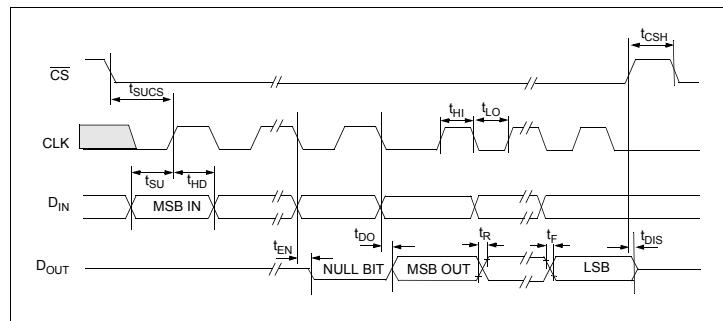


FIGURE 1-1: Serial Timing.

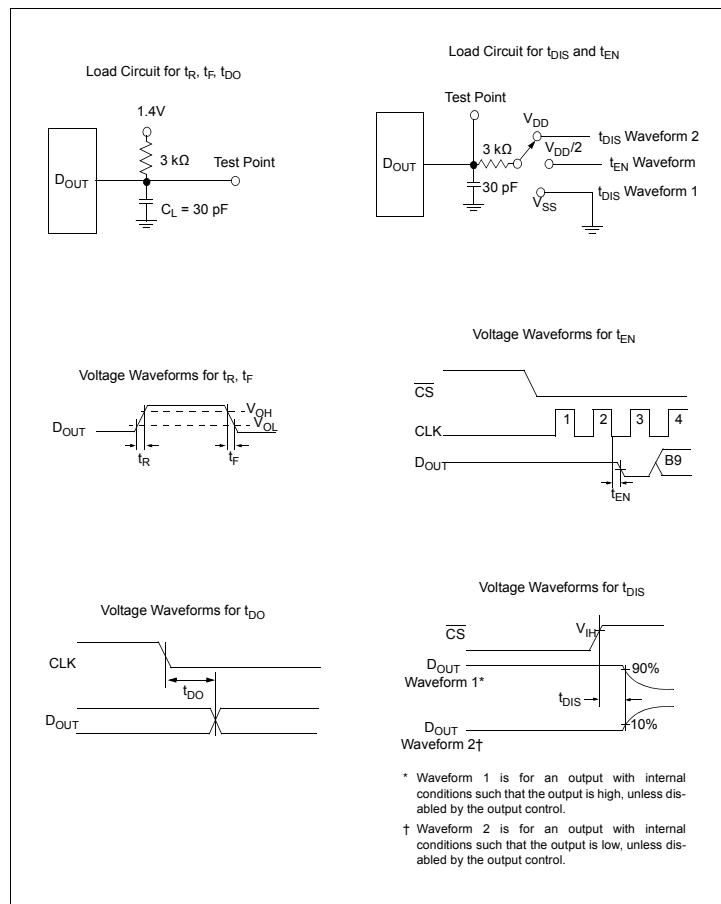
MCP3002

FIGURE 1-2: Test Circuits.

MCP3002

2.0 TYPICAL PERFORMANCE CHARACTERISTICS

Note: The graphs provided following this note are a statistical summary based on a limited number of samples and are provided for informational purposes only. The performance characteristics listed herein are not tested or guaranteed. In some graphs, the data presented may be outside the specified operating range (e.g., outside specified power supply range) and therefore outside the warranted range.

Note: Unless otherwise indicated, $V_{DD} = 5V$, $f_{SAMPLE} = 200$ kspS, $f_{CLK} = 16 \times f_{SAMPLE}$, $T_A = +25^\circ C$.

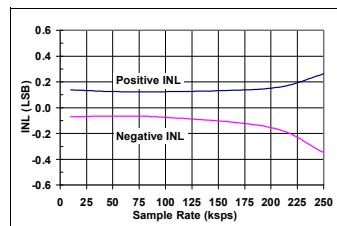


FIGURE 2-1: Integral Nonlinearity (INL) vs. Sample Rate.

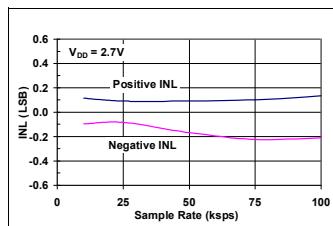


FIGURE 2-4: Integral Nonlinearity (INL) vs. Sample Rate ($V_{DD} = 2.7V$).

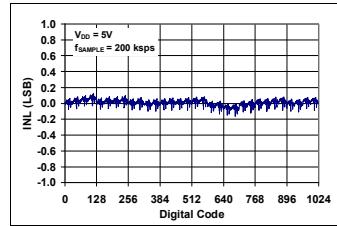


FIGURE 2-2: Integral Nonlinearity (INL) vs. Code.

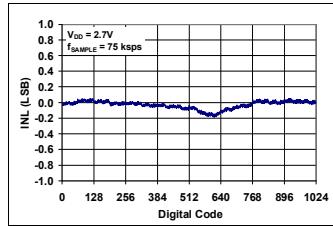


FIGURE 2-5: Integral Nonlinearity (INL) vs. Code ($V_{DD} = 2.7V$).

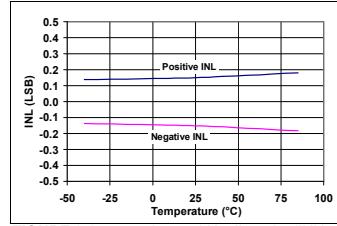


FIGURE 2-3: Integral Nonlinearity (INL) vs. Temperature.

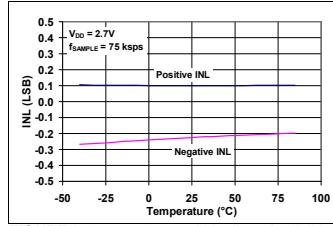


FIGURE 2-6: Integral Nonlinearity (INL) vs. Temperature ($V_{DD} = 2.7V$).

MCP3002

Note: Unless otherwise indicated, $V_{DD} = 5V$, $f_{SAMPLE} = 200$ ksp, $f_{CLK} = 16^* f_{SAMPLE}$, $T_A = +25^\circ C$.

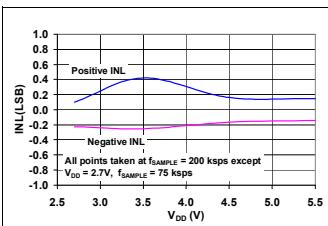


FIGURE 2-7: Integral Nonlinearity (INL) vs. V_{DD} .

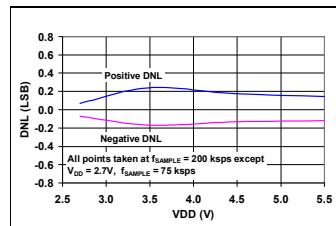


FIGURE 2-10: Differential Nonlinearity (DNL) vs. V_{DD} .

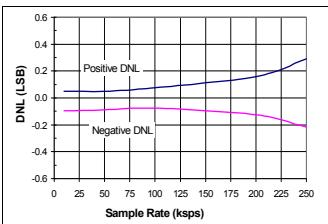


FIGURE 2-8: Differential Nonlinearity (DNL) vs. Sample Rate.

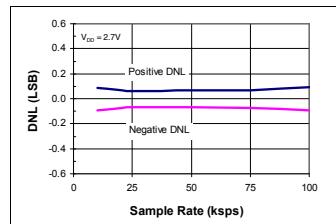


FIGURE 2-11: Differential Nonlinearity (DNL) vs. Sample Rate ($V_{DD} = 2.7V$).

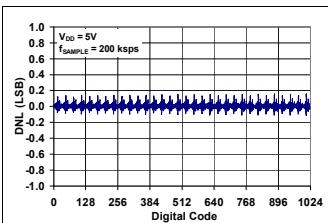


FIGURE 2-9: Differential Nonlinearity (DNL) vs. Code (Representative Part).

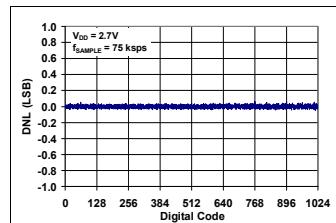


FIGURE 2-12: Differential Nonlinearity (DNL) vs. Code (Representative Part), $V_{DD} = 2.7V$.

MCP3002

Note: Unless otherwise indicated, $V_{DD} = 5V$, $f_{SAMPLE} = 200$ kspis, $f_{CLK} = 16 \times f_{SAMPLE}$, $T_A = +25^\circ C$.

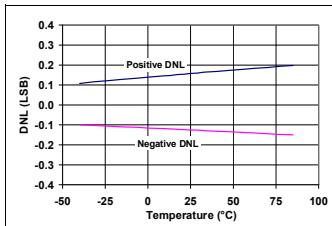


FIGURE 2-13: Differential Nonlinearity (DNL) vs. Temperature.

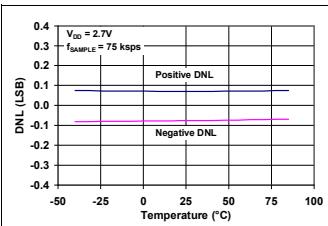


FIGURE 2-16: Differential Nonlinearity (DNL) vs. Temperature ($V_{DD} = 2.7V$).

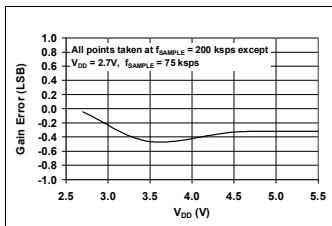


FIGURE 2-14: Gain Error vs. V_{DD} .

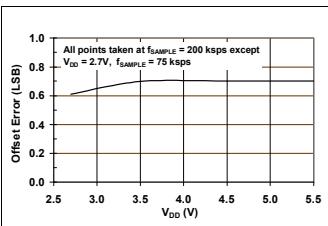


FIGURE 2-17: Offset Error vs. V_{DD} .

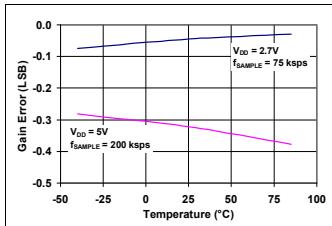


FIGURE 2-15: Gain Error vs. Temperature.

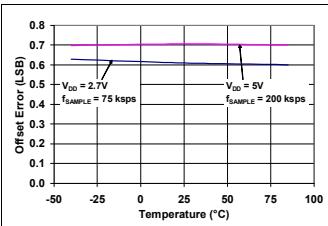


FIGURE 2-18: Offset Error vs. Temperature.

MCP3002

Note: Unless otherwise indicated, $V_{DD} = 5V$, $f_{SAMPLE} = 200$ ksp, $f_{CLK} = 16 \times f_{SAMPLE}$, $T_A = +25^\circ C$.

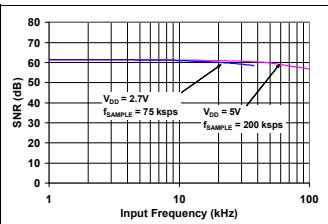


FIGURE 2-19: Signal-to-Noise Ratio (SNR) vs. Input Frequency.

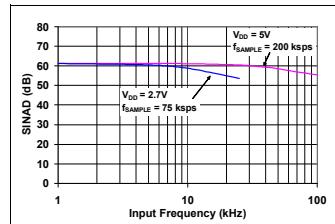


FIGURE 2-22: Signal-to-Noise and Distortion (SINAD) vs. Input Frequency.

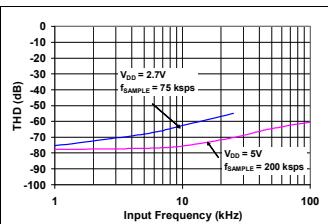


FIGURE 2-20: Total Harmonic Distortion (THD) vs. Input Frequency.

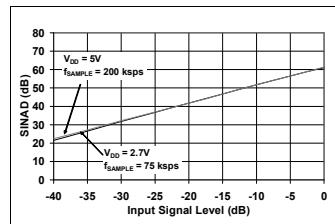


FIGURE 2-23: Signal-to-Noise and Distortion (SINAD) vs. Signal Level.

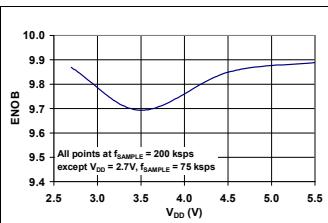


FIGURE 2-21: Effective Number of Bits (ENOB) vs. V_{DD} .

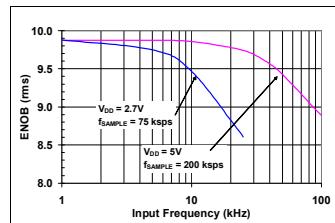


FIGURE 2-24: Effective Number of Bits (ENOB) vs. Input Frequency.

MCP3002

Note: Unless otherwise indicated, $V_{DD} = 5V$, $f_{SAMPLE} = 200$ ksp, $f_{CLK} = 16^* f_{SAMPLE}$, $T_A = +25^\circ C$.

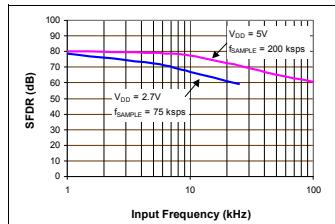


FIGURE 2-25: Spurious Free Dynamic Range (SFDR) vs. Input Frequency.

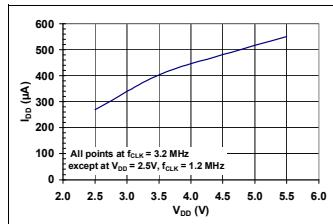


FIGURE 2-28: I_{DD} vs. V_{DD} .

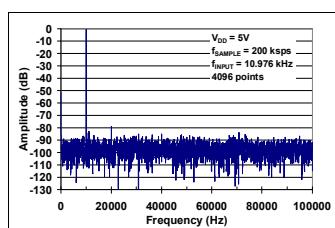


FIGURE 2-26: Frequency Spectrum of 10 kHz input (Representative Part).

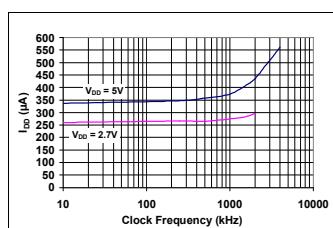


FIGURE 2-29: I_{DD} vs. Clock Frequency.

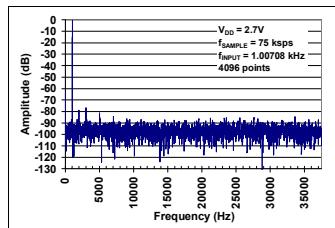


FIGURE 2-27: Frequency Spectrum of 1 kHz input (Representative Part, $V_{DD} = 2.7V$).

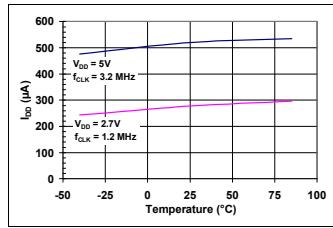


FIGURE 2-30: I_{DD} vs. Temperature.

MCP3002

Note: Unless otherwise indicated, $V_{DD} = 5V$, $f_{SAMPLE} = 200$ kspS, $f_{CLK} = 16 \times f_{SAMPLE}$, $T_A = +25^\circ C$.

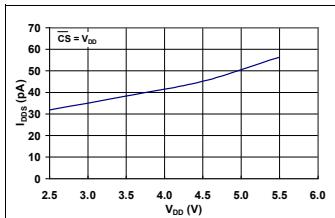


FIGURE 2-31: I_{DDS} vs. V_{DD} .

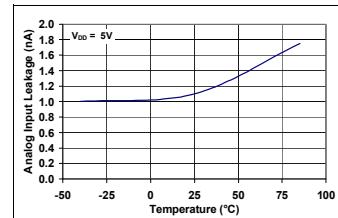


FIGURE 2-33: Analog Input leakage current vs. Temperature.

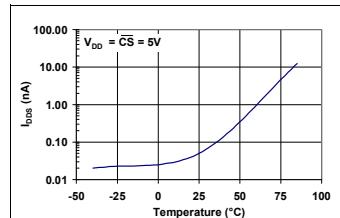


FIGURE 2-32: I_{DDS} vs. Temperature.

MCP3002

3.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in [Table 3-1](#). Additional descriptions of the device pins follows.

TABLE 3-1: PIN FUNCTION TABLE

MSOP, PDIP, SOIC, TSSOP	Name	Function
1	$\overline{\text{CS}}/\text{SHDN}$	Chip Select/Shutdown Input
2	CH0	Channel 0 Analog Input
3	CH1	Channel 1 Analog Input
4	V _{ss}	Ground
5	D _{IN}	Serial Data In
6	D _{OUT}	Serial Data Out
7	CLK	Serial Clock
8	V _{DD/V_{REF}}	+2.7V to 5.5V Power Supply and Reference Voltage Input

3.1 Analog Inputs (CH0/CH1)

Analog inputs for channels 0 and 1 respectively. These channels can be programmed to be used as two independent channels in Single-Ended mode or as a single pseudo-differential input where one channel is IN+ and one channel is IN-. See [Section 5.0 "Serial Communications"](#) for information on programming the channel configuration.

3.2 Chip Select/Shutdown ($\overline{\text{CS}}/\text{SHDN}$)

The $\overline{\text{CS}}/\text{SHDN}$ pin is used to initiate communication with the device when pulled low and will end a conversion and put the device in low power standby when pulled high. The $\overline{\text{CS}}/\text{SHDN}$ pin must be pulled high between conversions.

3.3 Serial Clock (CLK)

The SPI clock pin is used to initiate a conversion and to clock out each bit of the conversion as it takes place. See [Section 6.2 "Maintaining Minimum Clock Speed"](#) for constraints on clock speed.

3.4 Serial Data Input (D_{IN})

The SPI port serial data input pin is used to clock in input channel configuration data.

3.5 Serial Data Output (D_{OUT})

The SPI serial data output pin is used to shift out the results of the A/D conversion. Data will always change on the falling edge of each clock as the conversion takes place.

MCP3002

4.0 DEVICE OPERATION

The MCP3002 A/D converter employs a conventional SAR architecture. With this architecture, a sample is acquired on an internal sample/hold capacitor for 1.5 clock cycles starting on the second rising edge of the serial clock after the start bit has been received. Following this sample time, the input switch of the converter opens and the device uses the collected charge on the internal sample and hold capacitor to produce a serial 10-bit digital output code.

Conversion rates of 200 kspS are possible on the MCP3002. See [Section 6.2 "Maintaining Minimum Clock Speed"](#) for information on minimum clock rates.

Communication with the device is done using a 3-wire SPI-compatible interface.

4.1 Analog Inputs

The MCP3002 device offers the choice of using the analog input channels configured as two single-ended inputs that are referenced to V_{SS} or a single pseudo-differential input. The configuration setup is done as part of the serial command before each conversion begins. When used in the pseudo-differential mode, CH0 and CH1 are programmed as the IN+ and IN- inputs as part of the command string transmitted to the device. The IN+ input can range from IN- to the reference voltage, V_{DD} . The IN- input is limited to ± 100 mV from the V_{SS} rail. The IN- input can be used to cancel small signal common-mode noise which is present on both the IN+ and IN- inputs.

For the A/D converter to meet specification, the charge holding capacitor (C_{SAMPLE}) must be given enough time to acquire a 10-bit accurate voltage level during the 1.5 clock cycle sampling period. The analog input model is shown in [Figure 4-1](#).

In this diagram, it is shown that the source impedance (R_S) adds to the internal sampling switch (R_{SS}) impedance, directly affecting the time that is required to charge the capacitor, C_{SAMPLE} . Consequently, larger source impedances increase the offset, gain, and integral linearity errors of the conversion.

Ideally, the impedance of the signal source should be near zero. This is achievable with an operational amplifier such as the MCP601 which has a closed loop output impedance of tens of ohms. The adverse effects of higher source impedances are shown in [Figure 4-2](#).

When operating in the pseudo-differential mode, if the voltage level of IN+ is equal to or less than IN-, the resultant code will be 000h. If the voltage at IN+ is equal to or greater than [$V_{DD} + (IN-) - 1$ LSB], then the output code will be 3FFh. If the voltage level at IN- is more than 1 LSB below V_{SS} , then the voltage level at the IN+ input will have to go below V_{SS} to see the 000h output code. Conversely, if IN- is more than 1 LSB above V_{SS} , then the 3FFh code will not be seen unless the

IN+ input level goes above V_{DD} level. If the voltage at IN+ is equal to or greater than [$V_{DD} + (IN-) - 1$ LSB], then the output code will be 3FFh.

4.2 Digital Output Code

The digital output code produced by an A/D Converter is a function of the input signal and the reference voltage. For the MCP3002, V_{DD} is used as the reference voltage.

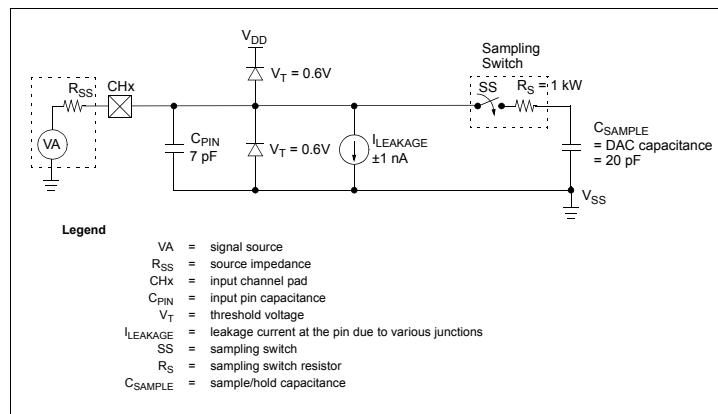
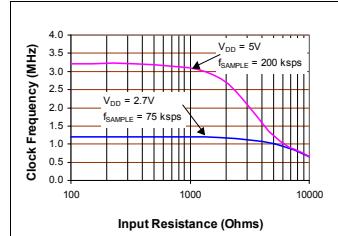
$$\text{LSB Size} = \frac{V_{REF}}{1024}$$

As the V_{DD} level is reduced, the LSB size is reduced accordingly. The theoretical digital output code produced by the A/D Converter is shown below.

$$\text{Digital Output Code} = \frac{1024 * V_{IN}}{V_{DD}}$$

Where:

$$\begin{aligned} V_{IN} &= \text{analog input voltage} \\ V_{DD} &= \text{supply voltage} \end{aligned}$$

MCP3002**FIGURE 4-1:** Analog Input Model.**FIGURE 4-2:** Maximum Clock Frequency vs. Input resistance (R_S) to maintain less than a 0.1 LSB deviation in INL from nominal conditions.

MCP3002

5.0 SERIAL COMMUNICATIONS

5.1 Overview

Communication with the MCP3002 is done using a standard SPI-compatible serial interface. Initiating communication with the device is done by bringing the CS line low. See [Figure 5-1](#). If the device was powered up with the CS pin low, it must be brought high and back low to initiate communication. The first clock received with CS low and D_{IN} high will constitute a start bit. The SGL/DIFF bit and the ODD/SIGN bit follow the start bit and are used to select the input channel configuration. The SGL/DIFF is used to select Single-Ended or Pseudo-Differential mode. The ODD/SIGN bit selects which channel is used in Single-Ended mode, and is used to determine polarity in Pseudo-Differential mode. Following the ODD/SIGN bit, the MSBF bit is transmitted to and is used to enable the LSB first format for the device. If the MSBF bit is high, then the data will come from the device in MSB first format and any further clocks with CS low, will cause the device to output zeros. If the MSBF bit is low, then the device will output the converted word LSB first after the word has been transmitted in the MSB first format. [Table 5-1](#) shows the configuration bits for the MCP3002. The device will begin to sample the analog input on the second rising edge of the clock, after the start bit has been received. The sample period will end on the falling edge of the third clock following the start bit.

On the falling edge of the clock for the MSBF bit, the device will output a low null bit. The next sequential 10 clocks will output the result of the conversion with MSB first as shown in [Figure 5-1](#). Data is always output from the device on the falling edge of the clock. If all 10 data bits have been transmitted and the device continues to receive clocks while the CS is held low (and the MSBF bit is high), the device will output the conversion result LSB first as shown in [Figure 5-2](#). If more clocks are provided to the device while CS is still low (after the LSB first data has been transmitted), the device will clock out zeros indefinitely.

If necessary, it is possible to bring CS low and clock in leading zeros on the D_{IN} line before the start bit. This is often done when dealing with microcontroller-based SPI ports that must send 8 bits at a time. Refer to [Section 6.1 "Using the MCP3002 with Microcontroller \(MCU\) SPI Ports"](#) for more details on using the MCP3002 devices with hardware SPI ports.

If it is desired, the CS can be raised to end the conversion period at any time during the transmission. Faster conversion rates can be obtained by using this technique if not all the bits are captured before starting a new cycle. Some system designers use this method by capturing only the highest-order 8 bits and 'throwing away' the lower 2 bits.

TABLE 5-1: CONFIGURING BITS FOR THE MCP3002

	CONFIG BITS		CHANNEL SELECTION		GND
	SGL/ DIFF	ODD/ SIGN	0	1	
Single-Ended Mode	1	0	+	—	—
	1	1	—	+	—
Pseudo- Differential Mode	0	0	IN+	IN-	—
	0	1	IN-	IN+	—

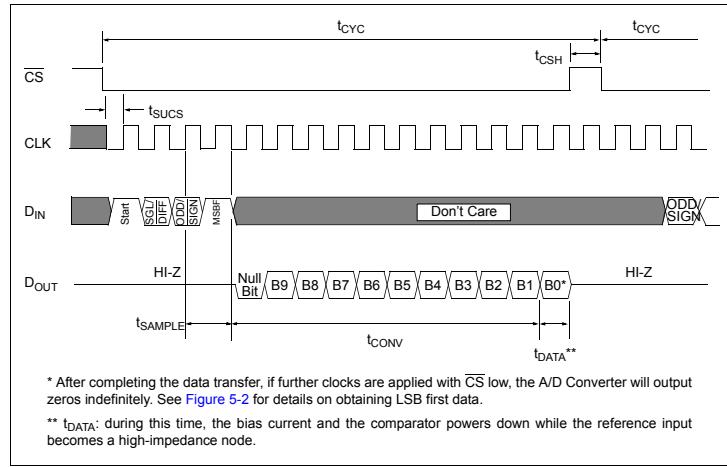
MCP3002

FIGURE 5-1: Communication with the MCP3002 using MSB first format only.

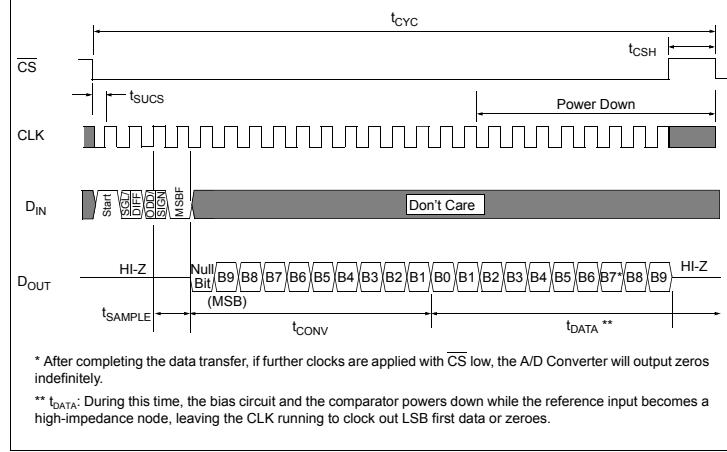


FIGURE 5-2: Communication with MCP3002 using LSB first format.

MCP3002

6.0 APPLICATIONS INFORMATION

6.1 Using the MCP3002 with Microcontroller (MCU) SPI Ports

With most microcontroller SPI ports, it is required to send groups of eight bits. It is also required that the microcontroller SPI port be configured to clock out data on the falling edge of clock and latch data in on the rising edge. Depending on how communication routines are used, it is very possible that the number of clocks required for communication will not be a multiple of eight. Therefore, it may be necessary for the MCU to send more clocks than are actually required. This is usually done by sending 'leading zeros' before the start bit, which are ignored by the device.

As an example, [Figure 6-1](#) and [Figure 6-2](#) show how the MCP3002 can be interfaced to a MCU with a hardware SPI port.

[Figure 6-1](#) depicts the operation shown in SPI Mode 0,0, which requires that the SCLK from the MCU idles in the 'low' state, while [Figure 6-2](#) shows the similar case of SPI Mode 1,1 where the clock idles in the 'high' state.

As shown in [Figure 6-1](#), the first byte transmitted to the A/D Converter contains one leading zero before the start bit. Arranging the leading zero this way produces the output 10 bits to fall in positions easily manipulated by the MCU. When the first 8 bits are transmitted to the device, the MSB data bit is clocked out of the A/D converter on the falling edge of clock number 6. After the second eight clocks have been sent to the device, the receive register will contain the lowest-order eight bits of the conversion results. Easier manipulation of the converted data can be obtained by using this method.

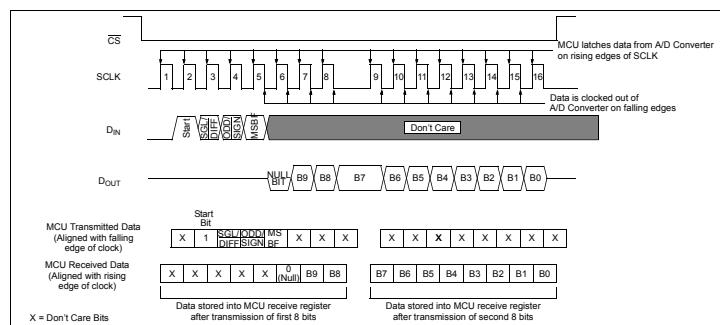


FIGURE 6-1: SPI Communication with the MCP3002 using 8-bit segments (Mode 0,0: SCLK idles low).

MCP3002

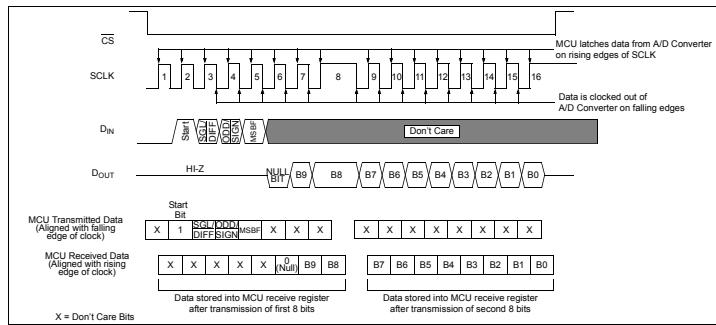


FIGURE 6-2: SPI Communication with the MCP3002 using 8-bit segments (Mode 1, 1: SCLK idles high).

6.2 Maintaining Minimum Clock Speed

When the MCP3002 initiates the sample period, charge is stored on the sample capacitor. When the sample period is complete, the device converts one bit for each clock that is received. It is important for the user to note that a slow clock rate will allow charge to bleed off the sample cap while the conversion is taking place. At 85°C (worst case condition), the part will maintain proper charge on the sample cap for 700 µs at $V_{DD} = 2.7V$ and 1.5 ms at $V_{DD} = 5V$. This means that at $V_{DD} = 2.7V$, the time it takes to transmit the 1.5 clocks for the sample period and the 10 clocks for the actual conversion must not exceed 700 µs. Failure to meet this criteria may induce linearity errors into the conversion outside the rated specifications.

6.3 Buffering/Filtering the Analog Inputs

If the signal source for the A/D Converter is not a low impedance source, it will have to be buffered or inaccurate conversion results may occur. It is also recommended that a filter be used to eliminate any signals that may be aliased back into the conversion results. This is illustrated in Figure 6-3 below where an op amp is used to drive, filter, and gain the analog input of the MCP3002. This amplifier provides a low impedance output for the converter input and a low-pass filter, which eliminates unwanted high-frequency noise.

Low-pass (anti-aliasing) filters can be designed using Microchip's interactive FilterLab® software. FilterLab will calculate capacitor and resistors values, as well as, determine the number of poles that are required for the application. For more information on filtering signals, see the application note AN699 "Anti-Aliasing Analog Filters for Data Acquisition Systems."

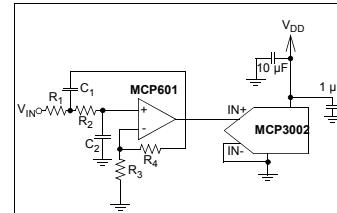


FIGURE 6-3: Typical Anti-Aliasing Filter Circuit (2 pole Active Filter).

MCP3002

6.4 Layout Considerations

When laying out a printed circuit board for use with analog components, care should be taken to reduce noise wherever possible. A bypass capacitor should always be used with this device and should be placed as close as possible to the device pin. A bypass capacitor value of $1\ \mu F$ is recommended.

Digital and analog traces should be separated as much as possible on the board and no traces should run underneath the device or the bypass capacitor. Extra precautions should be taken to keep traces with high-frequency signals (such as clock lines) as far as possible from analog traces.

Use of an analog ground plane is recommended in order to keep the ground potential the same for all devices on the board. Providing V_{DD} connections to devices in a "star" configuration can also reduce noise by eliminating current return paths and associated errors. See [Figure 6-4](#). For more information on layout tips when using A/D converters, refer to AN-688 "Layout Tips for 12-Bit A/D Converter Applications" (DS00688).

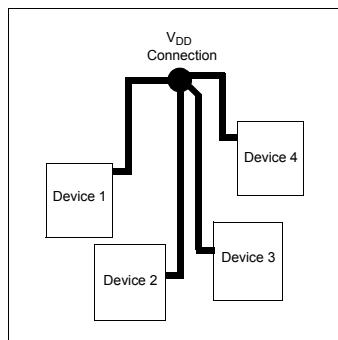


FIGURE 6-4: *V_{DD} traces arranged in a 'Star' configuration in order to reduce errors caused by current return paths.*

MCP3002

7.0 PACKAGING INFORMATION

7.1 Package Marking Information

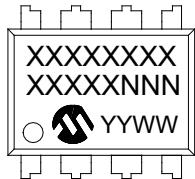
8-Lead MSOP (3x3 mm)



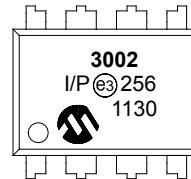
Example



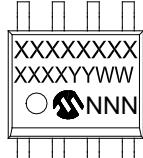
8-Lead PDIP (300 mil)



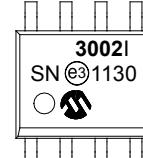
Example



8-Lead SOIC (3.90 mm)



Example



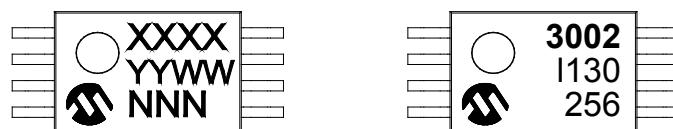
Legend:	XX...X	Customer specific information
Y	Year code (last digit of calendar year)	
YY	Year code (last 2 digits of calendar year)	
WW	Week code (week of January 1 is week '01')	
NNN	Alphanumeric traceability code	
(e3)	Pb-free JEDEC designator for Matte Tin (Sn)	
*	This package is Pb-free. The Pb-free JEDEC designator ((e3)) can be found on the outer packaging for this package.	

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line, thus limiting the number of available characters for customer-specific information.

MCP3002**Package Marking Information (Continued)**

8-Lead TSSOP (4.4 mm)

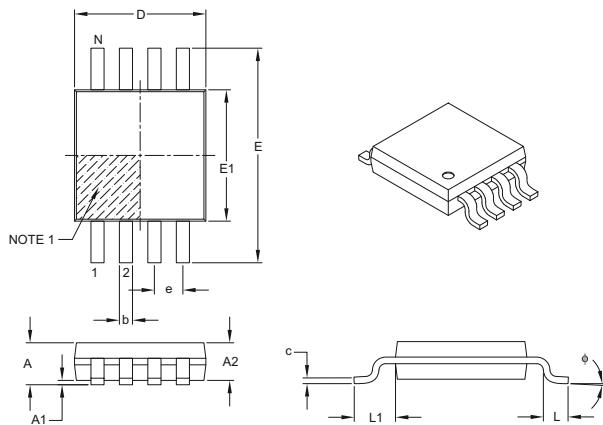
Example



Legend:	XX...X Customer-specific information
Y	Year code (last digit of calendar year)
YY	Year code (last 2 digits of calendar year)
WW	Week code (week of January 1 is week '01')
NNN	Alphanumeric traceability code
(e3)	Pb-free JEDEC designator for Matte Tin (Sn)
*	This package is Pb-free. The Pb-free JEDEC designator (e3) can be found on the outer packaging for this package.
Note:	In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line, thus limiting the number of available characters for customer-specific information.

MCP3002**8-Lead Plastic Micro Small Outline Package (MS) [MSOP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension	Limits	MILLIMETERS		
		MIN	NOM	MAX
Number of Pins	N	8		
Pitch	e		0.65 BSC	
Overall Height	A	—	—	1.10
Molded Package Thickness	A2	0.75	0.85	0.95
Standoff	A1	0.00	—	0.15
Overall Width	E		4.90 BSC	
Molded Package Width	E1		3.00 BSC	
Overall Length	D		3.00 BSC	
Foot Length	L	0.40	0.60	0.80
Footprint	L1		0.95 REF	
Foot Angle	ϕ	0°	—	8°
Lead Thickness	c	0.08	—	0.23
Lead Width	b	0.22	—	0.40

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

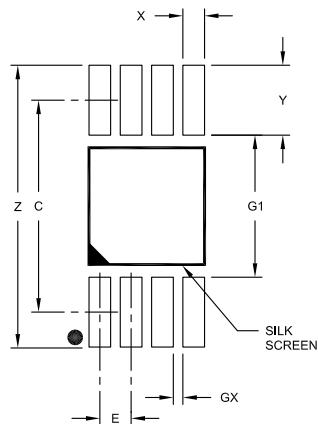
REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-111B

MCP3002

8-Lead Plastic Micro Small Outline Package (MS) [MSOP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at
<http://www.microchip.com/packaging>



RECOMMENDED LAND PATTERN

Dimension	Units	MILLIMETERS		
		MIN	NOM	MAX
Contact Pitch	E	0.65	BSC	
Contact Pad Spacing	C		4.40	
Overall Width	Z			5.85
Contact Pad Width (X8)	X1			0.45
Contact Pad Length (X8)	Y1			1.45
Distance Between Pads	G1	2.95		
Distance Between Pads	GX	0.20		

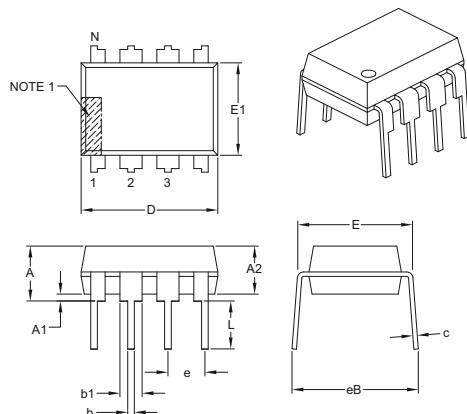
Notes:

1. Dimensioning and tolerancing per ASME Y14.5M
- BSC: Basic Dimension, Theoretically exact value shown without tolerances.

Microchip Technology Drawing No. C04-2111A

MCP3002**8-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension Limits	Units	INCHES		
	N	MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		.100 BSC	
Top to Seating Plane	A	—	—	.210
Molded Package Thickness	A2	.115	.130	.195
Base to Seating Plane	A1	.015	—	—
Shoulder to Shoulder Width	E	.290	.310	.325
Molded Package Width	E1	.240	.250	.280
Overall Length	D	.348	.365	.400
Tip to Seating Plane	L	.115	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.040	.060	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	—	—	.430

Notes:

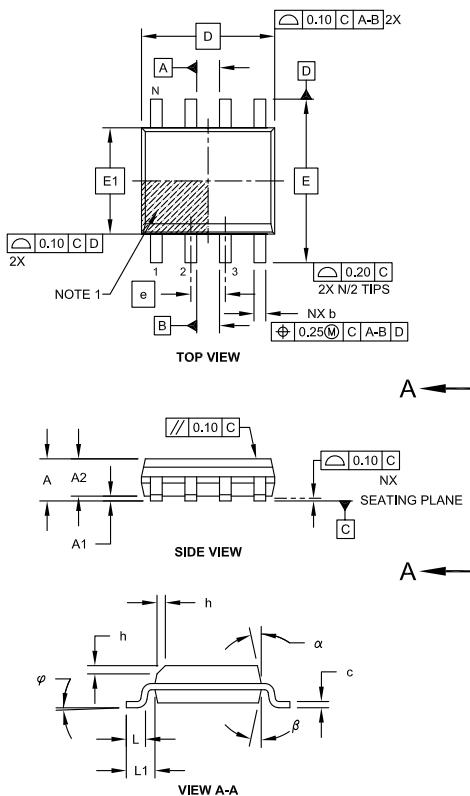
1. Pin 1 visual index feature may vary, but must be located with the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-018B

MCP3002**8-Lead Plastic Small Outline (SN) - Narrow, 3.90 mm Body [SOIC]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>

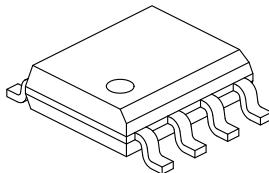


Microchip Technology Drawing No. C04-057C Sheet 1 of 2

MCP3002

8-Lead Plastic Small Outline (SN) - Narrow, 3.90 mm Body [SOIC]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension	Units	MILLIMETERS		
		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		1.27 BSC	
Overall Height	A	-	-	1.75
Molded Package Thickness	A2	1.25	-	-
Standoff	§	A1	0.10	-
Overall Width	E		6.00 BSC	
Molded Package Width	E1		3.90 BSC	
Overall Length	D		4.90 BSC	
Chamfer (Optional)	h	0.25	-	0.50
Foot Length	L	0.40	-	1.27
Footprint	L1		1.04 REF	
Foot Angle	φ	0°	-	8°
Lead Thickness	c	0.17	-	0.25
Lead Width	b	0.31	-	0.51
Mold Draft Angle Top	α	5°	-	15°
Mold Draft Angle Bottom	β	5°	-	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

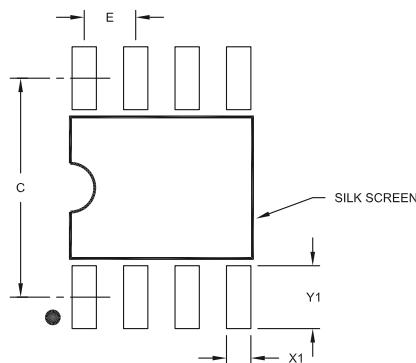
BSC: Basic Dimension, Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing No. C04-057C Sheet 2 of 2

MCP3002**8-Lead Plastic Small Outline (SN) – Narrow, 3.90 mm Body [SOIC]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



RECOMMENDED LAND PATTERN

Dimension	Units	MILLIMETERS		
		MIN	NOM	MAX
Contact Pitch	E		1.27 BSC	
Contact Pad Spacing	C		5.40	
Contact Pad Width (X8)	X1		0.60	
Contact Pad Length (X8)	Y1			1.55

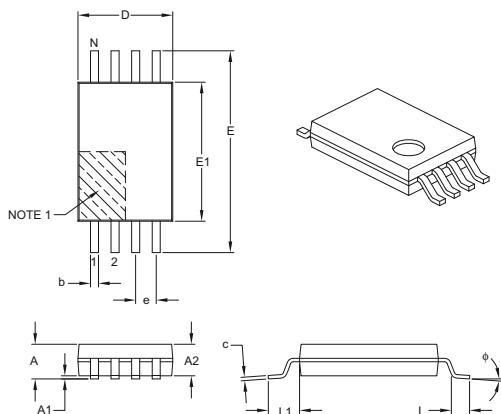
Notes:

- Dimensioning and tolerancing per ASME Y14.5M
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing No. C04-2057A

MCP3002**8-Lead Plastic Thin Shrink Small Outline (ST) – 4.4 mm Body [TSSOP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension	Limits	MILLIMETERS		
		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		0.65 BSC	
Overall Height	A	—	—	1.20
Molded Package Thickness	A2	0.80	1.00	1.05
Standoff	A1	0.05	—	0.15
Overall Width	E	6.40 BSC		
Molded Package Width	E1	4.30	4.40	4.50
Molded Package Length	D	2.90	3.00	3.10
Foot Length	L	0.45	0.60	0.75
Footprint	L1	1.00 REF		
Foot Angle	φ	0°	—	8°
Lead Thickness	c	0.09	—	0.20
Lead Width	b	0.19	—	0.30

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

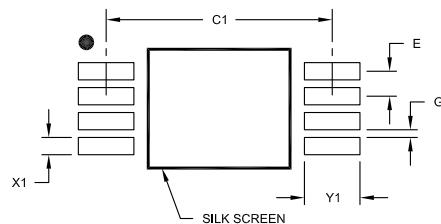
REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-086B

MCP3002

8-Lead Plastic Thin Shrink Small Outline (ST) - 4.4 mm Body [TSSOP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at
<http://www.microchip.com/packaging>



RECOMMENDED LAND PATTERN

Dimension Limits	Units	MILLIMETERS		
		MIN	NOM	MAX
Contact Pitch	E	0.65	BSC	
Contact Pad Spacing	C1		5.90	
Contact Pad Width (X8)	X1			0.45
Contact Pad Length (X8)	Y1			1.45
Distance Between Pads	G	0.20		

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing No. C04-2086A

MCP3002

APPENDIX A: REVISION HISTORY

Revision E (November 2011)

Updated [Product Identification System](#)

Corrected MSOP marking drawings.

Updated Package Specification Drawings with new additions.

Revision D (October 2008)

Updates to packaging outline drawings.

Revision C (January 2007)

Updates to packaging outline drawings.

Revision B (August 2001)

Undocumented changes.

Revision A (February 2000)

Initial release of this document.

MCP3002

PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

	<u>PART NO.</u>	X	XX	
Device	Device	Temperature Range	Package	
Device	MCP3002: 10-Bit Serial A/D Converter MCP3002T: 10-Bit Serial A/D Converter (Tape and Reel) (SOIC and TSSOP only)			Examples:
Temperature Range	I	= -40°C to +85°C (Industrial)		a) MCP3002-I/P: Industrial Temperature, 8LD PDIP package. b) MCP3002-I/SN: Industrial Temperature, 8LD SOIC package. c) MCP3002-I/ST: Industrial Temperature, 8LD TSSOP package. d) MCP3002-I/MS: Industrial Temperature, 8LD MSOP package.
Package	MS	= Plastic Micro Small Outline (MSOP), 8-lead		
	P	= Plastic DIP (300 mil Body), 8-lead		
	SN	= Plastic SOIC (150 mil Body), 8-lead		
	ST	= Plastic TSSOP (4.4 mm), 8-lead		

MCP3002

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² logo, rPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICWorks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MIWI, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omnisclient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SOTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2000-2011, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-61341-755-3

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMS, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
= ISO/TS 16949:2009 =**



Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://www.microchip.com/support>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada

Tel: 905-673-0699

Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong

Tel: 852-2401-1200

Fax: 852-2401-3431

Australia - Sydney

Tel: 61-2-9868-6733

Fax: 61-2-9868-6755

China - Beijing

Tel: 86-10-8569-7000

Fax: 86-10-8528-2104

China - Chengdu

Tel: 86-28-8665-5511

Fax: 86-28-8665-7889

China - Chongqing

Tel: 86-23-8980-9588

Fax: 86-23-8980-9500

China - Hangzhou

Tel: 86-571-2819-3187

Fax: 86-571-2819-3189

China - Hong Kong SAR

Tel: 852-2401-1200

Fax: 852-2401-3431

China - Nanjing

Tel: 86-25-8473-2460

Fax: 86-25-8473-2470

China - Qingdao

Tel: 86-532-8502-7355

Fax: 86-532-8502-7205

China - Shanghai

Tel: 86-21-5407-5533

Fax: 86-21-5407-5066

China - Shenyang

Tel: 86-24-2334-2829

Fax: 86-24-2334-2393

China - Shenzhen

Tel: 86-755-8203-2660

Fax: 86-755-8203-1760

China - Wuhan

Tel: 86-27-5980-5300

Fax: 86-27-5980-5118

China - Xian

Tel: 86-29-8833-7252

Fax: 86-29-8833-7256

China - Xiamen

Tel: 86-592-2388138

Fax: 86-592-2388130

China - Zhuhai

Tel: 86-756-3210040

Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune

Tel: 91-20-2666-1512
Fax: 91-20-2666-1513

Japan - Yokohama

Tel: 81-45-471-6166
Fax: 81-45-471-6122

Korea - Daegu

Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul

Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang

Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila

Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore

Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu

Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung

Tel: 886-7-536-4818
Fax: 886-7-330-9305

Taiwan - Taipei

Tel: 886-2-2500-6610
Fax: 886-2-2500-0120

Thailand - Bangkok

Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris

Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich

Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan

Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen

Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid

Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham

Tel: 44-118-921-5869
Fax: 44-118-921-5820

08/02/11

D.3 CAPACITIVE TOUCH IC DATASHEET



Atmel AT42QT1110-MZ AT42QT1110-AZ

11-key QTouch® Touch Sensor IC

DATASHEET

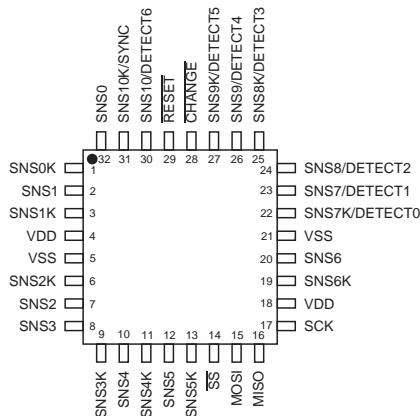
Features

- Sensor Keys:
 - Up to 11 QTouch® channels
- Data Acquisition:
 - Measurement of keys triggered either by a signal applied to the SYNC pin or at regular intervals timed by the AT42QT1110 internal clock
 - Keys measured sequentially for better performance, or in parallel groups for faster operation
 - Raw data for key touches can be read as a report over the SPI interface
- Discrete Outputs:
 - Configurable "Detect" outputs indicating individual key touch (7-key mode)
- Device Setup:
 - Device configuration can be stored in EEPROM
- Technology:
 - Patented spread-spectrum charge-transfer (direct mode)
- Key Outline Sizes:
 - 6 mm x 6 mm or larger (panel thickness dependent); widely different sizes and shapes possible, including solid or ring shapes
- Key Spacings:
 - 7 mm center-to-center or more (panel thickness dependent)
- Layers Required:
 - One
- Electrode Materials:
 - Etched copper, silver, carbon, Indium Tin Oxide (ITO)
- Electrode Substrates:
 - PCB, FPCB, plastic films, glass
- Panel Materials:
 - Plastic, glass, composites, painted surfaces (low particle density metallic paints possible)
- Panel Thickness:
 - Up to 10 mm glass, 5 mm plastic (electrode size dependent)
- Key Sensitivity:
 - Individually settable via simple commands over serial interface
- Adjacent Key Suppression® (AKS®)
 - Patented AKS technology to enable accurate key detection
- Interface:
 - Full-duplex SPI slave mode (1.5 MHz), CHANGE pin, discrete detection outputs

- Moisture Tolerance
 - Increased moisture tolerance based on hardware design and firmware tuning
- Power:
 - 3 V – 5.5 V
- Package:
 - 32-pin 5 × 5 mm MLF RoHS compliant
 - 32-pin 7 × 7 mm TQFP RoHS compliant
- Signal Processing:
 - Self-calibration, auto drift compensation, noise filtering, AKS technology
- Applications:
 - Specific package qualified for automotive applications, such as radio, keyless entry, electric windows and satellite navigation

1. Pinout and Schematic

1.1 Pinout Configuration



1.2 Pin Descriptions

Table 1-1. Pin Listing

Pin	Name	Type	Comments	If Unused, Connect To...
1	SNS0K	I/O	Sense Pin	Leave open
2	SNS1	I/O	Sense Pin	Leave open
3	SNS1K	I/O	Sense Pin	Leave open
4	Vdd	P	Power	—
5	Vss	P	Supply Ground	—
6	SNS2K	I/O	Sense Pin	Leave open
7	SNS2	I/O	Sense Pin	Leave open
8	SNS3	I/O	Sense Pin	Leave open
9	SNS3K	I/O	Sense Pin	Leave open
10	SNS4	I/O	Sense Pin	Leave open
11	SNS4K	I/O	Sense Pin	Leave open

Table 1-1. Pin Listing (Continued)

Pin	Name	Type	Comments	If Unused, Connect To...
12	SNS5	I/O	Sense Pin	Leave open
13	SNS5K	I/O	Sense Pin	Leave open
14	<u>SS</u>	I	Enable SPI	Vss via 100 kΩ resistor to enable SPI Vdd via 100 kΩ resistor to disable SPI
15	MOSI	I	SPI Data In	Leave open
16	MISO	O	SPI Data Out	Leave open
17	SCK	I	SPI Clock	Leave open
18	Vdd	P	Power	—
19	SNS6K	I/O	Sense Pin	Leave open
20	SNS6	I/O	Sense Pin	Leave open
21	Vss	P	Supply Ground	—
22	SNS7K/DETECT0	I/O	Sense Pin/Key Status Indicator	Leave open
23	SNS7/DETECT1	I/O	Sense Pin/Key Status Indicator	Leave open
24	SNS8/DETECT2	I/O	Sense Pin / Key Status Indicator	Leave open
25	SNS8K/DETECT3	I/O	Sense Pin / Key Status Indicator	Leave open
26	SNS9/DETECT4	I/O	Sense Pin / Key Status Indicator	Leave open
27	SNS9K/DETECT5	I/O	Sense Pin / Key Status Indicator	Leave open
28	<u>CHANGE</u>	OD	Touch Event Indicator	Leave open
29	<u>RESET</u>	I	Reset	Vdd
30	SNS10/DETECT6	I/O	Sense Pin / Key Status Indicator	Leave open
31	SNS10K/SYNC	I/O	Sense Pin / Synchronization Input	Vdd or Vss via 100 kΩ resistor
32	SNS0	I/O	Sense Pin	Leave open

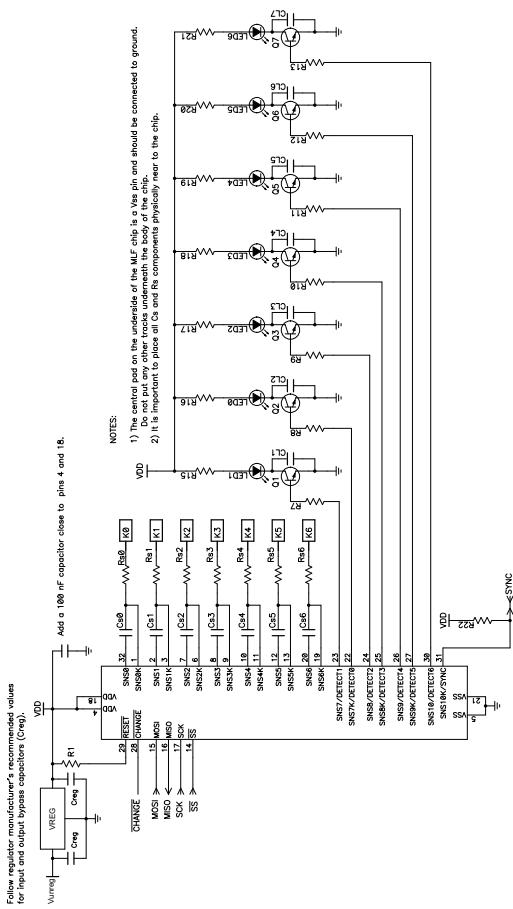
I Input only
 O Output only, push-pull

I/O Input and output
 OD Open drain output

P Ground or power

1.3 Schematics

Figure 1-1. Typical Circuit: 7 keys With Detect Outputs and No External Trigger





Follow regulator manufacturer's recommended values for input and output bypass capacitors (Creg).

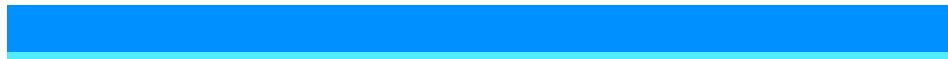
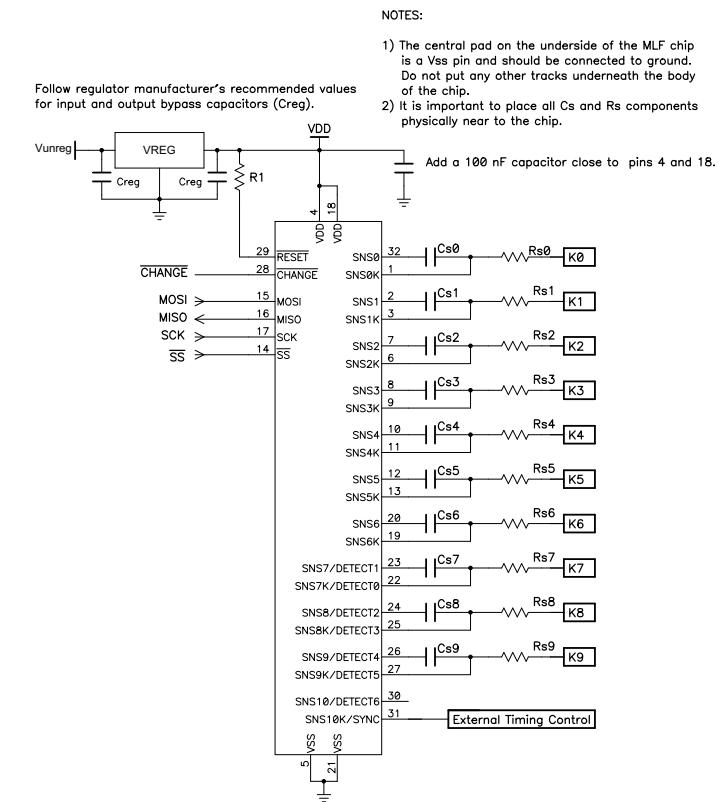


Figure 1-3. Typical Circuit: 10 Keys With External Trigger (SYNC Mode)



For component values in Figure 1-1, Figure 1-2 and Figure 1-3, check the following sections:

- [Section 3.1 on page 9](#): Cs capacitors (Cs0 – Cs10)
- [Section 3.2 on page 9](#): Sample resistors (Rs0 – Rs10)
- [Section 3.5 on page 10](#): Voltage levels
- [Section 3.3 on page 9](#): LED traces

2. Overview of the AT42QT1110

2.1 Introduction

The AT42QT1110 (QT1110) is a digital burst mode charge-transfer (QTTM) capacitive sensor driver designed for any touch-key applications.

The keys can be constructed in different shapes and sizes. Refer to the *Touch Sensors Design Guide* and Application Note QTAN0002, *Secrets of a Successful QTouch Design*, for more information on construction and design methods (both downloadable from the Atmel website).

The device includes all signal processing functions necessary to provide stable sensing under a wide variety of changing conditions, and the outputs are fully debounced. Only a few external parts are required for operation.

The QT1110 modulates its bursts in a spread-spectrum fashion in order to suppress heavily the effects of external noise, and to suppress RF emissions.

2.2 Configurations

The QT1110 is designed as a versatile device, capable of various configurations. There are two basic configurations for the QT1110:

- 11-key QTouch. The device can sense up to 11 keys.
- 7-key QTouch with individual outputs for each key. The device can sense up to 7 keys and drive the matching Detect outputs to a user-configurable PWM.

Both configurations allow for a choice of acquisition modes, thus providing a variety of possibilities that will satisfy most applications (see the following sections for more information).

Additionally, the SYNC line can be used as an external trigger input. Note that in 11-key mode the SYNC line replaces one key, thus allowing only 10 keys.

See [Section 4.7 on page 18](#) for more information.

2.3 Guard Channel

The device has a guard channel option (available in all key modes), which allows one key to be configured as a guard channel to help prevent false detection. See [Section 4.9 on page 20](#) for more information.

2.4 Self-test Functions

The QT1110 has two types of self-test functions:

- Internal Hardware tests – check for hardware failures in the device internal memory.
- Functional checks – confirm that the device is operating within expected parameters.

See [Section 4.10 on page 20](#) for more information.

2.5 Moisture Tolerance

The presence of water (condensation, sweat, spilt water, and so on) on a sensor can alter the signal values measured and thereby affect the performance of any capacitive device. The moisture tolerance of QTouch devices can be improved by designing the hardware and fine-tuning the firmware following the recommendations in the application note Atmel AVR3002: *Moisture Tolerant QTouch Design* (www.atmel.com/Images/doc42017.pdf).

3. Wiring and Parts

3.1 Cs Sample Capacitors

Cs0 – Cs10 are the charge sensing sample capacitors. Normally they are identical in nominal value. The optimal Cs values depend on the thickness of the panel and its dielectric constant. Thicker panels require larger values of Cs. Values can be in the range 2.2 nF (for faster operation) to 33 nF (for best sensitivity); typical values are 4.7 nF to 10 nF.

The value of Cs should be chosen so that a light touch on a key produces a reduction of ~20 to 30 in the key signal value (see [Section 6.8 on page 27](#)). The chosen Cs value should never be so large that the key signals exceed ~1000, as reported by the chip in the debug data.

The Cs capacitors must be X7R or PPS film type, for stability. For consistent sensitivity, they should have a 10 percent tolerance. Twenty percent tolerance may cause small differences in sensitivity from key to key and unit to unit. If a key is not used, the Cs capacitor may be omitted.

3.2 Rs Resistors

The series resistors Rs0 – Rs10 are inline with the electrode connections and should be used to limit electrostatic discharge (ESD) currents and to suppress radio frequency (RF) interference. Values should be approximately 2 k Ω to 20 k Ω each; a typical value is 4.7 k Ω .

Although these resistors may be omitted, the device may become susceptible to external noise or radio frequency interference (RFI). For details of how to select these resistors see the Application Note QTAN0002, *Secrets of a Successful QTouch Design*, downloadable from the Touch Technology area of the Atmel website, www.atmel.com.

3.3 LED Traces and Other Switching Signals

Digital switching signals near the sense lines can induce transients into the acquired signals, deteriorating the SNR performance of the device. Such signals should be routed away from the sensing traces and electrodes, or the design should be such that these lines are not switched during the course of signal acquisition (bursts).

LED terminals which are multiplexed or switched into a floating state, and which are within, or physically very near, a key (even if on another nearby PCB) should be bypassed to either Vss or Vdd with at least a 1 nF capacitor. This is to suppress capacitive coupling effects which can induce false signal shifts. The bypass capacitor does not need to be next to the LED, in fact it can be quite distant. The bypass capacitor is noncritical and can be of any type.

LED terminals which are constantly connected to Vss or Vdd do not need further bypassing.

3.4 PCB Cleanliness

Modern no-clean flux is generally compatible with capacitive sensing circuits.



CAUTION: If a PCB is reworked to correct soldering faults relating to the QT1110, or to any associated traces or components, be sure that you fully understand the nature of the flux used during the rework process. Leakage currents from hygroscopic ionic residues can stop capacitive sensors from functioning. If you have any doubts, a thorough cleaning after rework may be the only safe option.

3.5 Power Supply

3.5.1 General Considerations

See [Section 8.2 on page 39](#) for the power supply range. If the power supply fluctuates slowly with temperature, the device tracks and compensates for these changes automatically with only minor changes in sensitivity. If the supply voltage drifts or shifts quickly, the drift compensation mechanism is not able to keep up, causing sensitivity anomalies or false detections.

The usual power supply considerations with QT parts apply to the device. The power should be clean and come from a separate regulator if possible. However, this device is designed to minimize the effects of unstable power, and, except in extreme conditions, should not require a separate Low Dropout (LDO) regulator.

See underneath [Figure 1.3 on page 5](#) for suggested regulator manufacturers.



Caution: A regulator IC shared with other logic can result in erratic operation and is **not** advised.

A single ceramic 0.1 μF bypass capacitor, with short traces, should be placed very close to the power pins of the IC. Failure to do so can result in device oscillation, high current consumption, or erratic operation.

It is assumed that a larger bypass capacitor (like 1 μF) is somewhere else in the power circuit; for example, near the regulator.

3.5.2 Brownout Detection

The QT1110 includes a power supply monitoring circuit that detects if Vdd drops below a safe operating voltage. When this occurs, the device goes into a **Reset** state, where no acquisition or processing is carried out. The device remains in this state until Vdd returns to the specified voltage range.

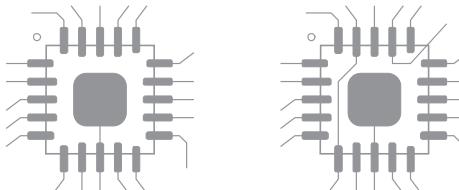
Once a safe operating voltage is detected, the QT1110 behaves as per normal power-on/reset conditions; that is, any saved settings are restored from EEPROM, the internal self-tests are run and all channels are calibrated.

The Brown-out detector threshold is 2.7 V $\pm 10\%$.

3.6 MLF Package Restrictions

The central pad on the underside of the MLF chip should be connected to ground. Do not run any tracks underneath the body of the chip, only ground. [Figure 3-1](#) shows examples of good and bad tracking.

Figure 3-1. Examples of Good and Bad Tracking



Example of GOOD tracking

Example of BAD tracking

4. Detailed Operations

4.1 Communications

4.1.1 Introduction

All communication with the device is carried out over the Serial Peripheral Interface (SPI). This is a synchronous serial data link that operates in full-duplex mode. The host communicates with the QT controller over the SPI using a master-slave relationship, with the QT1110 acting in slave mode.

4.1.2 SPI Operation

The SPI uses four logic signals:

- Serial Clock (SCK) – output from the host.
- Master Output, Slave Input (MOSI) – output from the host, input to the QT controller. Used by the host to send data to the QT controller.
- Master Input, Slave Output (MISO) – input to the host, output from the QT controller. Used by the QT device to send data to the host.
- Slave Select (\overline{SS}) – active low output from the host.

At each byte, the master pulls \overline{SS} low and generates 8 clock pulses on SCK. With these 8 clock pulses, a byte of data is transmitted from the master to the slave over MOSI, most significant bit (msb) first.

Simultaneously a byte of data is transmitted from the slave to the master over MISO, also most significant bit first.

The slave reads the status of MOSI at the leading edge of each clock pulse, and the master reads the slave data from MISO at the trailing edge.

The QT1110 requires that the clock idles "high", meaning that the data on MOSI and MISO pins are set at the falling edges and sampled at the rising edges.

That is:

Clock polarity CPOL = 1
Clock phase CPHA = 1

The QT1110 SPI interface can operate at any SCK frequency up to 1.5 MHz.

In multibyte communications, the master must pause for a minimum delay of 150 μ s between the completion of one byte exchange and the beginning of the next.

Note that the number of bytes to be transmitted depends on the initial command sent by the host. This sets the mode on the QT1110 so that the QT1110 knows how to respond to, or how to interpret, the following bytes. If there is a delay of >100 ms between bytes while the QT1110 is waiting for data, or waiting to send data, then the incomplete transmission is discarded and the device resets its SPI state machine. It will then interpret the next byte it receives as a fresh command.

When the QT1110 SPI interface is receiving a new command, it returns the *Idle* status code (0x55) on MISO during the first byte exchange to indicate to the master that it is in the correct state for receiving instructions.

4.1.3 CRC Bytes

If enabled, a CRC checking procedure is implemented on all communications between the SPI master and the QT1110. In this case, each command or report request sent by the master must have a byte appended containing the CRC checksum of the data sent. The QT1110 will not respond to commands until the CRC byte has been received and verified.

Sample C code showing the algorithm for calculating the CRC of the data can be found in [Appendix A](#).

When the QT1110 is expecting a CRC byte, it returns (on MISO) the calculated CRC byte which it expects to receive. This is sent simultaneously with the QT1110 receiving the CRC byte from the master (that is, during the same byte exchange). This allows both devices to confirm that the data was sent correctly.

All data returned by the QT1110 is also followed by a CRC byte, allowing the master to confirm the integrity of the data transmission.

4.1.4 SPI Commands

There are three types of communication between the SPI master and the QT1110:

- Control commands (see [Section 5. on page 22](#))
 - To send control instructions to the QT1110
- Report requests (see [Section 6. on page 25](#))
 - To reading status information from the QT1110
- Setup commands (see [Section 7. on page 29](#))
 - To set configuration options ("Set" instructions)
 - To read configuration options ("Get" instructions)

Additionally the NULL command (0x00) is transmitted by the host device as it is receiving data from the QT1110.

4.1.4.1 Control Commands

A control command is an instruction sent to the QT1110 that controls operations of the device, and for which no response is required. Examples of control commands are: *Reset*, *Calibrate*, *Send Setups*.

With the exception of *Send Setups*, control commands normally require a single byte exchange, unless CRC checking is enabled, in which case a second byte must be transmitted by the host with the calculated CRC of the command byte.

Figure 4-1. Sleep Command – CRC Disabled

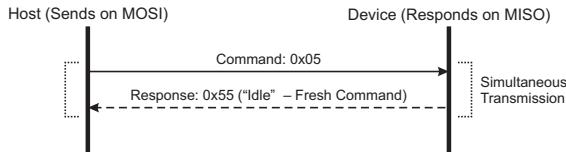
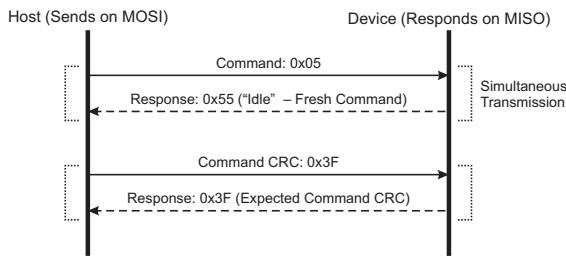


Figure 4-2. Sleep Command – CRC Enabled



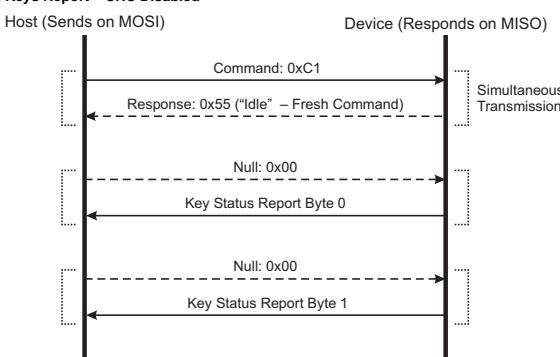
When the *Send Setups* command is received, the QT1110 stops measurement of QTouch sensors and waits for 42 bytes of data to be sent. Only when all 42 bytes have been received (and the CRC byte, if CRC is enabled), the QT1110 applies all the settings to RAM and resumes measurement. In this case, if CRC is enabled, the CRC byte is calculated for all the data sent by the host, including the command byte 0x01.

Control Commands are specified in detail in [Section 5. on page 22](#).

4.1.5 Report Requests

Report Requests are sent by the Host to instruct the QT1110 to return status information. The host sends the appropriate *Report Request* command, then transmits Null bytes on MOSI while the QT1110 returns the report data on MISO.

Figure 4-3. All Keys Report – CRC Disabled



For example, [Figure 4-3](#) shows the exchange that takes place to read the 2-byte *All Keys* report. In this exchange, the host sends:

0xC1 — 0x00 — 0x00

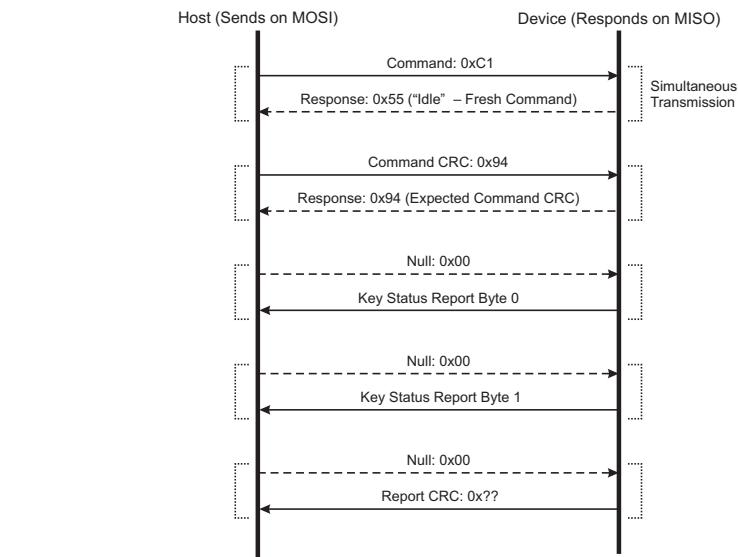
and the QT1110 returns (simultaneously):

0x55 — Report Byte 0 — Report Byte 1

If CRC is enabled, this exchange is extended to 5 bytes, as shown in [Figure 4-4 on page 14](#).



Figure 4-4. All Keys Report – CRC Enabled



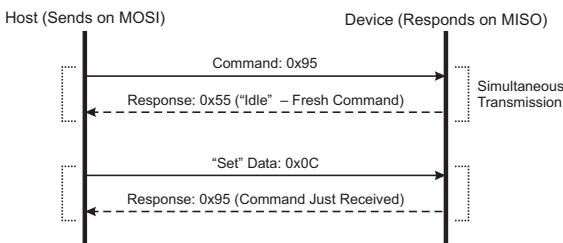
4.1.5.1 Set Instructions

Set Instructions are 2-byte transmissions by the host that are used to send settings to individual locations in the device memory map.

At the first byte, the QT1110 returns 0x55 (*Idle*) to confirm that it will interpret the byte as a new command. At the second byte, the QT1110 returns the *Set* command it has just received.

For example, to set the *Positive Recalibration Delay* to 1920 ms, address 5 in the memory map is set to 12 (0x0C). This is done with the *Set* command for address 5 (command code 0x95), as shown in Figure 4-5.

Figure 4-5. Positive Recalibration Delay Set Instruction – CRC Disabled

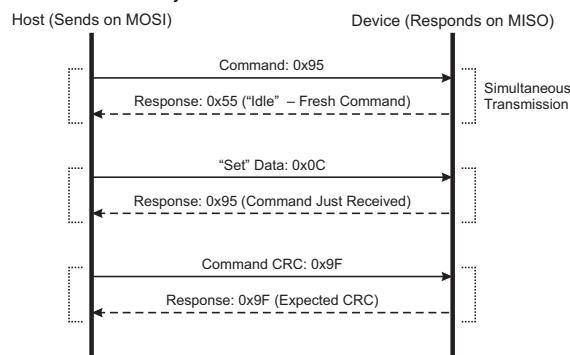


With CRC Enabled, a CRC byte is also required (Figure 4-6). This is calculated for the two transmitted bytes (that is, the Set command and the data byte).

For example, for the sequence shown in Figure 4-5 (0x95 — 0x0C), the CRC Byte is 0x9F. As is the case with the other command types, when the QT1110 is expecting a CRC byte from the host, it calculates that byte in advance and returns the expected value to the host in the same transmission as the host sends the CRC byte.

The sent data is not applied to the memory location until the CRC byte has been received and verified.

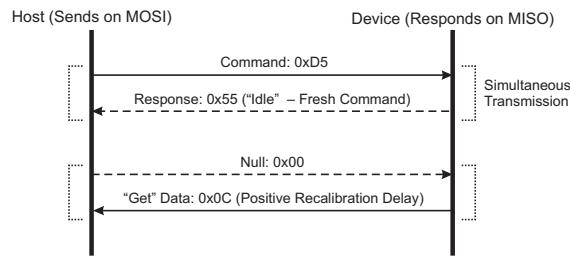
Figure 4-6. Positive Recalibration Delay Set Instruction – CRC Enabled



4.1.5.2 Get Instructions

Get instructions are instructions that read the data from a location in the QT1110 memory map.

Figure 4-7. Positive Recalibration Delay Get Instruction – CRC Disabled



The host sends the appropriate Get command, followed by a Null byte. The QT1110 returns the contents of the addressed memory location.

Figure 4-7 shows the exchange for a report on the positive recalibration delay (assuming that the data byte is 0x0C).

With CRC Enabled, this exchange takes 4 bytes, with a command CRC transmitted by the host and a report CRC returned by the QT1110 (see Figure 4-8 on page 16).

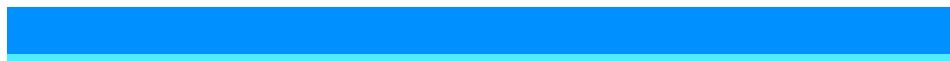
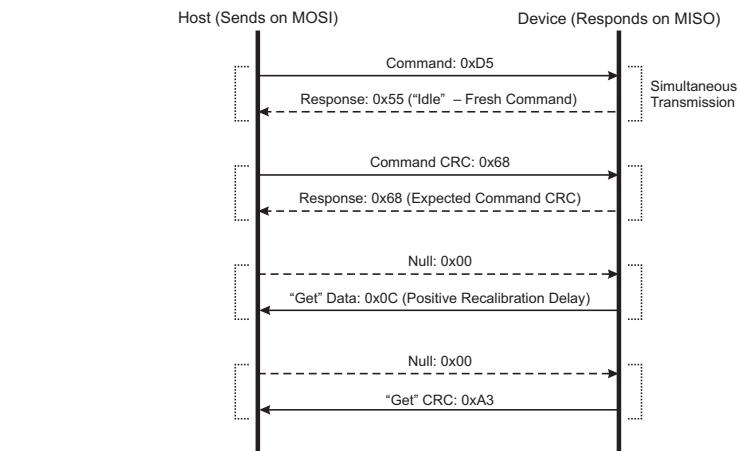


Figure 4-8. Positive Recalibration Delay Get Instruction – CRC Enabled



4.1.6 Quick SPI Mode

4.1.6.1 Introduction

In Quick SPI Mode, the QT1110 sends a 7-byte key report at each exchange. No host commands are required over SPI in this mode; the host clocks the data bytes out in sequence. Quick SPI mode is enabled by setting the *SPI_EN* bit in the Comms Options setup byte (see [Section 7.5 on page 31](#)).

4.1.6.2 Quick SPI Report

The 7 report bytes are in the format given in [Table 4-1](#).

Table 4-1. Device Status Report Format

Byte	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Counter	Counter – increments from 0 to 255							
1	Detect status, channels 0 – 3	Channel 3	Channel 2	Channel 1	Channel 0				
2	Detect status, channels 4 – 7	Channel 7	Channel 6	Channel 5	Channel 4				
3	Detect status, channels 8 – 10	Reserved	Channel 10	Channel 9	Channel 8				
4	Error status, channels 0 – 3	Channel 3	Channel 2	Channel 1	Channel 0				
5	Error status, channels 4 – 7	Channel 7	Channel 6	Channel 5	Channel 4				
6	Error status, channels 8 – 10	Reserved	Channel 10	Channel 9	Channel 8				

where:

- Byte 0 is a counter that increments from 0 to 254 on successive exchanges to confirm that firmware is operating correctly.
- Bytes 1 – 3 indicate the detect status of channels 0 – 3, 4 – 7 and 8 – 10 respectively (two bits per channel), as follows:
 - 00 = Channel not in detect
 - 01 = Channel in detect
 - 10 = Not Allowed
 - 11 = Invalid Signal (Channel disabled)
- Bytes 4 – 6 indicate the error status of channels 0 – 3, 4 – 7 and 8 – 10 respectively (two bits per channel), as follows:
 - 00 = No error
 - 01 = Not allowed
 - 10 = Error on channel
 - 11 = Invalid signal (channel disabled)

Successive byte exchanges in Quick SPI mode cycle through the 7 bytes of status information. If synchronization is lost, the host must either re-synchronize by identifying the incrementing counter byte (byte 0) or pausing communications for at least 100 ms so the QT1110 will reset its SPI state.

4.1.6.3 Commands in Quick SPI Mode

Only two host commands are recognized under Quick SPI mode. These are shown in [Table 4-2](#).

Table 4-2. Host Commands in Quick SPI Mode

Command	Code	Purpose
Store to EEPROM	0x0A	Allows for "Quick SPI mode" to be stored as the default start-up mode
Enable Full SPI	0x36	Enables full SPI mode

CRC checking is not implemented in Quick SPI mode for host commands or return data.

4.1.6.4 Quick SPI Mode timing

In Quick SPI mode, the minimum time between byte exchanges is reduced to 50 µS.

If a pause in communications of 100 ms is detected during reading of the 7-byte report, the QT1110 resets the exchange, and on the next byte read it returns byte 0 of the report.

4.2 Reset

The QT1110 can be reset using one of two methods:

- **Hardware reset:** An external reset logic line can be used if desired, fed into the RESET pin. However, under most conditions it is acceptable to tie RESET to Vdd.
- **Software reset:** A software reset can be forced using the "Reset" control command.

For both methods, the device will follow the same initialization sequence. If there any saved settings in the EEPROM, these are loaded into RAM. Otherwise the default settings are applied.

Note: The SPI interface becomes active after the QT1110 has completed its startup sequence, taking approximately 160 ms after power on/reset.

4.3 Sleep Mode

The QT1110 can be put into a very low power sleep mode (typically $< 2 \mu\text{A}$). During sleep mode, no keys are measured and the DETECT outputs are all put into high impedance mode to minimize current consumption. The device remains in sleep mode until a falling edge is detected on either the SS pin or the CHANGE pin. When the QT1110 wakes from sleep mode, it continues to operate as it was before it was put into sleep mode. The QT1110 requires approximately 100 μs to wake from sleep mode and will not respond correctly to SPI communications until the wake-up procedure is complete. The low level on the SS or CHANGE pin that is used to wake the device must be maintained for 100 μs to ensure correct operation.

Note: If the device is set to sleep mode for an extended period, the host should initiate a recalibration immediately after waking the QT1110.

4.4 Calibration

The device can be forced to recalibrate the sensor keys at any time. This can be useful where, for example, a portable device is plugged into mains power, or during product development when settings are being tuned.

The QT1110 can also be configured to automatically recalibrate if it remains in detection for too long. This avoids keys becoming "stuck" after a prolonged period of uninterrupted detection. See [Section 7.18 on page 38](#) for details.

4.5 CHANGE Pin

The CHANGE pin can be configured using the Comms Options setup byte (see [Section 7.5 on page 31](#)) to act in one of two modes:

- Data mode
 - The CHANGE pin is asserted (pulled low) when the detection status of a key changes from that last sent to the host; that is when a key-touch or key-release event occurs.
 - The CHANGE pin is pulled low when a key status changes and is only released when the "Send All keys" report is requested (0xC1), or the key status information bytes are read in Quick SPI mode (see [Section 7.5 on page 31](#)).
- Touch mode
 - The CHANGE pin is pulled low when one or more keys are in detect. The CHANGE pin remains low as long as there is a key in detect, regardless of communications.
 - The CHANGE pin is released when there are no keys in detect. No host communications are required to release the CHANGE pin.

4.6 Stand-alone Mode

The QT1110 can operate in a stand-alone mode without the use of the SPI interface. The settings are loaded from EEPROM and the device operates in 7-key mode using the Detect outputs.

4.7 Key Modes

4.7.1 11-key Mode

In 11-key mode, the device can sense up to 11 keys. Alternatively, one key can be replaced by the SYNC line as an external trigger input (see [Section 4.8.2 on page 19](#)).

11-key mode is configured by setting the MODE bit in the Device Mode setup byte (see [Section 7.4 on page 30](#)).

Key acquisition can be triggered in one of two ways: using the internal clock to trigger acquisition either at a fixed repetition period or in a continuous "free run" mode (see [Section 4.8.1](#)), or using the SYNC pin to provide an external trigger (see [Section 4.8.2 on page 19](#)),

4.7.2 7-key Mode

In 7-key mode, the detect outputs DETECT0 to DETECT6 become active on pins 22 – 27 and 30. These outputs provide configurable PWM signals that indicate when each of the keys is touched.

7-key mode is configured by clearing the *MODE* bit in the Device Mode setup byte (see [Section 7.4 on page 30](#)).

Each DETECT output can be individually configured to output a PWM signal while the matching key is in detect or out of detect. This signal can be one of nine levels, ranging from low (PWM = 0%) to high (PWM = 100%). This allows for the use of an indicating LED. This is achieved by enabling the appropriate bit in the Key to LED setup byte (see [Section 7.14 on page 36](#)), and setting the desired outputs levels or PWMs in setup addresses 9 to 15 (see [Section 7.12 on page 34](#)).

4.8 Trigger Modes

4.8.1 Timed Trigger

In 11-key mode, The QT1110 can be configured to use the internal clock as a timed trigger. In this case, the QT1110 is configured with a cycle period, such that each acquisition cycle starts a specified length of time after the start of the previous cycle. If the cycle period is set to 0, each acquisition cycle starts as soon as the previous one has finished, resulting in the acquisition cycles running back-to-back in a “free run” mode.

The use of a timed trigger, and the cycle period to be used, is set in the Device Mode setup byte (see [Section 7.4 on page 30](#)).

4.8.2 Synchronized Trigger

In 11-key mode, if a time trigger is not enabled, the QT1110 operates in “synchronized” mode. In this mode, SNS10K is used as a SYNC pin to trigger key acquisition, rather than using the device internal clock. In this case the maximum number of keys is reduced to 10.

The SYNC pin can use one of two methods to trigger key measurements, selectable via bit 4 of the Device Mode setup byte (see [Section 7.4 on page 30](#)): Low Level and Rising Edge.

With the Low Level method the QT1110 operates in “free run” mode for as long as the SYNC pin is read as a logical 0. When the SYNC pin goes high, the current measurement cycle will be finished and no more key measurements will be taken until the SYNC pin goes low again. The low level trigger should be a minimum of 1 ms so that there is sufficient time for the device to detect the low level.

With the Rising Edge method all enabled keys are measured once when a rising edge is detected on the SYNC pin. This allows key measurements to be synchronized to an external event or condition.

For example, the SYNC pin can be used by the host to synchronize several devices to each other. This would ensure that only one of the devices outputs pulses at any given time and signals from one QT1110 do not interfere with the measurements from another.

Another use for synchronizing to the rising edge is to steady the signals when the device is running off a mains transformer with insufficient mains frequency filtering that is causing a 50 Hz or 60 Hz ripple on Vdd. If the mains voltage is scaled down with a simple voltage divider and connected to the SYNC pin, then the key measurement can be triggered by the rising edge detected at a positive going zero-crossing. Note that in this case, each key signal will be taken at the same point in the cycle, so Vdd will be the same at each measurement for a given key and the signals will be steadier.



4.9 Guard Channel Option

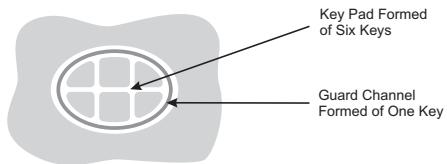
The device has a guard channel option (available in all key modes), which allows one key to be configured as a guard channel to help prevent false detection (see [Figure 4-9 on page 20](#)). Guard channel keys should be more sensitive than the other keys (physically bigger or larger Cs), subject to burst length limitations (see [Section 4.11.2 on page 21](#)).

With guard channel enabled, the designated key is connected to a sensor pad which detects the presence of touch and overrides any output from the other keys using the chip AKS feature. The guard channel option is enabled by the Guard Key setup byte (see [Section 7.5 on page 31](#)).

With the guard channel not enabled, all the keys work normally.

Note: If a key is already "in detect" when the guard channel becomes active, that key will remain in detect and the guard key will not activate until the active key goes out of detect.

Figure 4-9. Guard Channel Example



4.10 Self-test Functions

4.10.1 Internal Hardware Tests

Internal hardware tests check for hardware failure in the device internal memory areas and data paths. Any failure detected in the function or contents of application ROM, RAM or registers causes the device to reset itself.

The application code is scanned with a CRC check routine to confirm that the application data is all correct.

The RAM and registers are checked periodically (every 10 seconds) for dynamic and static failures.

4.10.2 Functional Checks

Functional checks confirm that the device is operating within expected parameters; any failure detected in these tests is notified to the system host. The device will continue to operate in the event that such functional failures are detected.

The functional tests are:

- Check that the channel-measurement signals are within the defined range.
- Confirm that data stored in the EEPROM is valid.

These tests are carried out as the particular functions are used. For example, the EEPROM is checked when the device attempts to load data from EEPROM, and the channel signals are checked when a measurement is carried out.

Note: If a particular channel is unused, the threshold of that channel should be set to 0 to prevent the incorrect reporting of the unused channel as being in an error state.

4.11 Signal Processing

4.11.1 Detection Integrator

The device features a detection integration mechanism, which acts to confirm a detection in a robust fashion. A per-key counter is incremented each time the key has exceeded its threshold. When this counter reaches a preset limit the key is finally declared to be touched. For example, if the DI limit is set to 10, then a key signal must fall by more than the key threshold, and remain below that level for 10 acquisitions, before the key is declared to be touched.

Similarly, the DI is applied to a key that is going out of detect: it must take 10 acquisitions where the signal has not exceeded its detect threshold before it is declared to leave touch.

4.11.2 Burst Length Limitations

The maximum burst length is 2048 pulses. The recommended design is to use a capacitor that gives a signal of <1000 pulses.

The number of pulses in the burst can be obtained by reading the key signal (that is, the number of pulses to complete measurement of the key signal) over the SPI interface (see [Section 6.8 on page 27](#)). Alternatively, a scope can be used to measure the entire burst, and then the burst length divided by the time for a single pulse.

Note that the keys are independent of each other. It is therefore possible, for example, to have a signal of 100 on one key and a signal of 1000 on another.

4.11.3 Adjacent Key Suppression Technology

The device includes the Atmel patented Adjacent Key Suppression (AKS) technology to allow the use of tightly spaced keys on a keypad with no loss of selectability by the user.

AKS is enabled or disabled for each key individually; only one key out of those enabled for AKS may be reported as touched at any one time. The first key touched dominates and stays in detect until it is released, even if another stronger key is reported. Once it is released, the next strongest key is reported. If two keys are simultaneously detected, the strongest key is reported, allowing a user to slide a finger across multiple keys with only the dominant key reporting touch.

Each key can be enabled for AKS processing via the AKS mask (see [Section 7.11 on page 34](#)). Keys outside the group of enabled keys may be in detect simultaneously.

5. Control Commands

5.1 Introduction

The QT1110 control commands are those commands that affect the device operation.

The control commands are listed in [Table 5-1](#) and are described individually in the following sections.

Table 5-1. Control Commands

Command	Code	Note
Send Setups	0x01	Configures the device to receive setup data
Calibrate All	0x03	Calibrates all keys
Reset	0x04	Resets the device
Sleep	0x05	Sleep (dead) mode
Store to EEPROM	0x0A	Stores RAM setups to EEPROM
Restore from EEPROM	0x0B	Copies EEPROM setups to RAM (automatically done at startup)
Erase EEPROM	0x0C	Erases EEPROM setups
Recover EEPROM	0x0D	Restores last EEPROM settings (after erase)
Calibrate Key k	0x1k	Calibrates one key (key k)

Note: Commands are implemented immediately upon reception, so a suitable delay is required for the operation to be completed before communications can be re-established.

5.2 Send Setups (0x01)

This command initiates the upload of the full settings table to the QT1110 (see [Section 7. on page 29](#)).

When this command is received, the QT1110 stops key measurement and waits until 42 bytes of setup data have been received. Key acquisition will restart after all the setup data has been received.

If enabled, a CRC check byte is transmitted (both ways) after the 42 bytes to confirm that they have been received correctly.

If CRC checking is not enabled, it is recommended that the host request a dump of setup data from the QT1110, and confirms that the data correctly matches the data sent.

The host must wait for at least 150 μ s for the operation to be completed before communications can be re-established.

5.3 Calibrate All (0x03)

This command initiates the recalibration of all sensor keys.

The host must wait for at least 150 μ s for the operation to be completed before communications can be re-established.

5.4 Reset (0x04)

The Reset command forces the QT1110 to reset. If the setups data is present in the EEPROM, the setups are loaded into the device. Otherwise default settings are applied.

The host must wait for at least 160 ms for the operation to be completed before communications can be re-established.

5.5 Sleep (0x05)

The Sleep command puts the device into sleep mode (see [Section 4.3 on page 18](#)).

The host must wait for at least 150 μ s after a low signal is applied to the \overline{SS} or \overline{CHANGE} pin to wake the device before communications can be re-established.

5.6 Store to EEPROM (0x0A)

Stores the current RAM contents to the QT1110 internal EEPROM. When the device is reset, it will automatically reload these settings.

The host must wait for at least 200 ms for the operation to be completed before communications can be re-established.

5.7 Restore from EEPROM (0x0B)

Settings stored in EEPROM are automatically loaded into RAM when the device is reset. If desired, these settings can be re-loaded into RAM using the *Restore from EEPROM* command.

The host must wait for at least 150 ms for the operation to be completed before communications can be re-established.

5.8 Erase EEPROM (0x0C)

This command erases the settings stored in EEPROM and then resets the QT1110. This causes the QT1110 to revert to its default settings.

The host must wait for at least 50 ms for the operation to be completed before communications can be re-established.

5.9 Recover EEPROM (0x0D)

This command "undeletes" the setup data that was previously stored in the device EEPROM and has been erased using the "Erase EEPROM" command.

Note: If valid settings have not previously been stored in the device EEPROM, the QT1110 continues to operate under the default settings.

The host must wait for at least 50 ms for the operation to be completed before communications can be re-established.

5.10 Calibrate Key (0x1k)

This command recalibrates the key specified by k . For example, to calibrate key 4, the host sends 0x14; to calibrate key 10, the host sends 0x1A.

The host must wait for at least 150 μ s for the operation to be completed before communications can be re-established.



6. Report Requests

6.1 Introduction

The host can request reports from the QT1110, as summarized in [Table 6-1](#).

Table 6-1. Report Requests

Command	Code	Note	Data Returned
Send First Key	0xC0	Returns the first detected key	1 byte
Send All keys	0xC1	Returns all keys	2-byte bitfield
Device Status	0xC2	Returns the device status	1-byte bitfield
EEPROM CRC	0xC3	Returns the EEPROM CRC	1 byte
RAM CRC	0xC4	Returns the RAM CRC	1 byte
Error Keys	0xC5	Returns the error keys	2-byte bitfield
Signal for Key <i>k</i>	0x2k	Returns the signal for key <i>k</i>	2-byte number
Reference for Key <i>k</i>	0x4k	Returns the reference for key <i>k</i>	2-byte number
Status for Key <i>k</i>	0x8k	Returns error conditions/touch indication for key <i>k</i>	1 byte
Detect Output States	0xC6	Returns the detect output states	1 byte
Last Command	0xC7	Returns the last command sent to QT1110	1 byte
Setups	0xC8	Returns the setup data	42 bytes
Device ID	0xC9	Returns the device ID	1 byte
Firmware Version	0xCA	Returns the firmware version	1 byte

Note that SPI communications are full-duplex, so the host must transmit on the MOSI pin to keep the communications active, while reading data from the QT1110 on the MOSI pin. Failure to do this within 100 ms will cause the device to assume that the exchange has been abandoned and reset the SPI interface. The host should therefore send one or two "NULL" bytes, as appropriate, on the MOSI line as it receives the 1- or 2-byte report data from the device.

6.2 First Key (0xC0)

This command returns 1-byte report in the format shown in [Table 6-2](#).

Table 6-2. Send First Key Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	DETECT	NUMKEY	ERROR				KEY_NUM	

DETECT: 0 = no key in detect; 1 = there is a key in detect.

NUMKEY: indicates the number of keys in detect:

0 = only one key is in detect (specified by "KEY_NUM")

1 = more than one key in detect.

ERROR: 0 = there are no keys in an error state; 1 = at least one key is in error state.
KEY_NUM: the key number (0 to 10) of the key in detect (if there is only one), or the number of the first key to go into detection when there are more than one.

6.3 All Keys (0xC1)

Returns a 2-byte bit-field report indicating the detection status of all 11 keys.

Table 6-3. Send All Keys Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0						KEY_10	KEY_9	KEY_8
Byte 1	KEY_7	KEY_6	KEY_5	KEY_4	KEY_3	KEY_2	KEY_1	KEY_0

KEY_n: 0 = key *n* out of detect, 1 = key *n* in detect (where *n* is 0 – 10).

6.4 Device Status (0xC2)

This command returns a 1-byte bit-field report indicating the overall status of the QT1110.

Table 6-4. Device Status Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	1	DETECT	CYCLE	ERROR	CHANGE	EEPROM	RESET	GUARD

Bits 7 is always 1; the other bits are as follows:

DETECT: 0 = no key in detect, 1 = at least 1 key in detect.

CYCLE: 0 = cycle time is good, 1 = cycle time over-run. A cycle time over-run occurs when it takes longer to measure and process all the keys than the assigned cycle time.

ERROR: 0 = no key in error state, 1 = at least 1 key in error.

CHANGE: 0 = CHANGE pin is asserted, 1 = CHANGE pin is floating.

EEPROM: 0 = EEPROM is good, 1 = EEPROM has an error. If there are no settings stored in EEPROM, the EEPROM error bit is set and a zero EEPROM CRC is returned.

RESET: set to 1 after power-on or reset, cleared when "Device Status" is read.

GUARD: 0 = guard channel is not in detect, 1 = guard channel is active or in detect. This bit will be zero if the guard channel is not enabled.

6.5 EEPROM CRC (0xC3)

This command returns a 1-byte CRC checksum for the setup data in EEPROM.

6.6 RAM CRC (0xC4)

This command returns a 1-byte CRC checksum for the setup data in RAM.



6.7 Error Keys (0xC5)

This command returns a 2-byte bit-field report indicating the error status of all 11 keys. Note that disabled keys do not report errors.

Table 6-5. Send All Keys Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0						KEY_10	KEY_9	KEY_8
Byte 1	KEY_7	KEY_6	KEY_5	KEY_4	KEY_3	KEY_2	KEY_1	KEY_0

KEY_n: 0 = key *n* status good, 1 = key *n* in error (where *n* is 0–10).

6.8 Signal for Key *k* (0x2k)

This command returns a 2-byte report containing the most recent measured signal for key *k*. The signal is returned as a 16-bit number, MSB first.

Table 6-6. Signal for Key *k* Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0						Signal MSB		
Byte 1						Signal LSB		

6.9 Reference for Key *k* (0x4k)

This command returns a 2-byte report containing the reference signal for key *k*. The reference is returned as a 16-bit number, MSB first.

Table 6-7. Reference for Key *k* Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0						Reference MSB		
Byte 1						Reference LSB		

6.10 Status for Key *k* (0x8k)

This command returns a 1-byte report containing the status for key *k*.

Table 6-8. Status for Key *k* Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	DETECT	LBL	MBL			AKS_EN	CAL	KEY_EN

DETECT: Out of detect, 1 = in detect.

LBL: 0 = lower burst limit is good, 1 = lower burst limit has error.

MBL: 0 = maximum burst limit is good, 1 = maximum burst limit has error. The maximum burst limit is fixed at 2048 pulses.

AKS_EN: 0 = AKS is disabled, 1 = AKS is enabled.

CAL: 0 = normal, 1 = calibrating.

KEY_EN: 0 = key is disabled, 1 = key is enabled.

6.11 Detect Output States (0xC6)

This command returns a byte that indicates which PWM signal is applied to each DETECT pin.

Table 6-9. Detect Output States

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0		DET_6	DET_5	DET_4	DET_3	DET_2	DET_1	DET_0

DET_n: 0 = "Out of Detect" PWM is output, 1 = the "In Detect" PWM is output.

Note: Note: During "LED Detect Hold Time" or "LED Fade", the report indicates the new state of the DETECT pin. For example, if the DETECT output is in "LED Detect Hold Time" before switching to "Out of Detect" PWM, the reported state is "0".

6.12 Last Command (0xC7)

This command returns the previous 1-byte command that was received from the host. Note that this command does not return itself.

Table 6-10. Last Command

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0								Last Command

6.13 Setups (0xC8)

This command returns the 42 bytes of the setups table, starting with address 0, with the most significant bit first.

6.14 Device ID (0xC9)

This command returns 1 byte containing the device ID (0x57).

Table 6-11. Device ID Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0								Device ID = 0x57

6.15 Firmware Version (0xCA)

Returns 1 byte containing the firmware version.

Table 6-12. Firmware Version Report Format

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0								Major Version

7. Setups and Status Information

7.1 Introduction

The bytes of the setup table can be written to or read from individually. The setup table and the corresponding *Set* and *Get* commands are listed in [Table 7-1](#). Note that there is a discontinuity in the *Set* and *Get* commands; 0xAF and 0xEF are not implemented.

Table 7-1. Memory Map

Address	Function	Set Command	Get Command
0	Device Mode	0x90	0xD0
1	Guard Key/Comms Options	0x91	0xD1
2	Detect Integrator (DI)/Drift Hold Time (DHT)	0x92	0xD2
3	Positive Threshold (PTHR)/Positive Hysteresis (PHYST)	0x93	0xD3
4	Positive Drift Compensation (PDRIFT)	0x94	0xD4
5	Positive Recalibration Delay (PRD)	0x95	0xD5
6	Lower Burst Limit (LBL)	0x96	0xD6
7	AKS Mask: Keys 8–10	0x97	0xD7
8	AKS Mask: Keys 0–7	0x98	0xD8
9	Detect0 PWM “Detect”/PWM “No Detect”	0x99	0xD9
10	Detect1 PWM “Detect”/PWM “No Detect”	0x9A	0xDA
11	Detect2 PWM “Detect”/PWM “No Detect”	0x9B	0xDB
12	Detect3 PWM “Detect”/PWM “No Detect”	0x9C	0xDC
13	Detect4 PWM “Detect”/PWM “No Detect”	0x9D	0xDD
14	Detect5 PWM “Detect”/PWM “No Detect”	0x9E	0xDE
15	Detect6 PWM “Detect”/PWM “No Detect”	0x9F	0xDF
16	LED Detect Hold Time	0xA0	0xE0
17	LED Fade/Key to LED	0xA1	0xE1
18	LED Latch	0xA2	0xE2
19	Key0 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA3	0xE3
20	Key1 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA4	0xE4
21	Key2 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA5	0xE5
22	Key3 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA6	0xE6
23	Key4 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA7	0xE7
24	Key5 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA8	0xE8
25	Key6 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xA9	0xE9
26	Key7 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xAA	0xEA
27	Key8 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xAB	0xEB

Table 7-1. Memory Map (Continued)

Address	Function	Set Command	Get Command
28	Key9 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xAC	0xEC
29	Key10 Negative Threshold (NTHR)/Negative Hysteresis (NHYST)	0xAD	0xED
30	Extend Pulse Time	0xAE	0xEE
31	Key0 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB0	0xF0
32	Key1 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB1	0xF1
33	Key2 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB2	0xF2
34	Key3 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB3	0xF3
35	Key4 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB4	0xF4
36	Key5 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB5	0xF5
37	Key6 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB6	0xF6
38	Key7 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB7	0xF7
39	Key8 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB8	0xF8
40	Key9 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xB9	0xF9
41	Key10 Negative Drift Compensation (NDRIFT)/Negative Recalibration Delay (NRD)	0xBA	0xFA

7.2 Setting Individual Settings

To set up an individual setup value, the host sends the command listed under the "Set Command" column in [Table 7-1](#), followed by a byte of data.

For details of the communication flow, see [Section 4.1 on page 11](#).

7.3 Setting All the Setups

The host can send all 42 bytes of setup data to the QT1110 as a block using the Send Setups command. See [Section 5.2 on page 22](#) for details.

7.4 Address 0: Device Mode

The Device Mode controls the overall operation of the device: number of keys, acquisition method, timing and trigger mechanism.

Table 7-2. Device Mode

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	KEY_AC	MODE	SIGNAL	SYNC			REPEAT_TIME	

KEY_AC: selects the trigger source to start key acquisition; 0 = SYNC pin, 1 = timed.

MODE: selects 7-key or 11-key mode; 0 = default 7-key mode, 1 = 11-key mode.

SIGNAL: selects serial or parallel acquisition of keys signals; 0 = serial, 1 = parallel.

SYNC: selects the trigger type when SYNC Pin is selected as the trigger to start key acquisition.

0 = Level Acquisition starts when a 0 is read at the SYNC pin. If the pin is held low, the QT1110 operates in *Free run* mode (that is, it will not sleep in between acquisitions, but start again immediately).

1 = Edge Acquisition starts when a rising edge is detected at the SYNC pin. When acquisition and post-processing are completed, the device sleeps until another rising edge is detected at the SYNC pin.

REPEAT_TIME: selects the “repeat” time when “Timed” is selected as the trigger to start key acquisition. The number entered is a multiple of 16 ms. If 0 is entered, the device will operate in a continuous *free run* mode; that is, the QT1110 will not sleep after its cycle is completed but will begin the next key acquisition cycle immediately.

Default KEY_AC value: 1 (timed)
Default MODE value: 0 (7-key mode)
Default SIGNAL value: 1 (parallel)
Default SYNC value: 1 (edge)
Default REPEAT_TIME value: 2 (32 ms cycle)

7.5 Address 1: Guard Key/Comms Options

Table 7-3. Guard Key/Comms Options

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1			GUARD_KEY		GD_EN	SPI_EN	CHG	CRC

GUARD_KEY: specifies the key (0 to 10) to be used as a guard channel (see [Section 2.3 on page 8](#)).

GD_EN: enables the use of a guard key; 0 = disable, 1 = enable.

SPI_EN: enables the Quick SPI interface; 0 = disable, 1 = enable.

See [Section 4.1.6 on page 16](#) for details of the Quick SPI Mode report.

To exit this mode (and clear the **SPI_EN** bit), the command 0x36 should be sent. To save the settings to EEPROM and make Quick SPI mode active on startup, send the *Store to EEPROM* command (0x0A). Any other data sent is ignored in Quick SPI mode.

CHG: the **CHANGE** pin mode (see [Section 4.5 on page 18](#)):

0 = *Data* mode. In this mode the **CHANGE** pin is asserted to indicate unread data.

1 = *Touch* mode. In this mode the **CHANGE** pin is asserted when a key is being touched or is in detect.

CRC: enables or disables CRC; 0 = disable, 1 = enable. When this option is enabled, each data exchange must have a CRC byte appended.

When report or setup data is being returned by the QT1110, a 1-byte checksum is returned. The host should confirm that this checksum is correct and, if not, should request the report again.

Where data is being sent by the host, a 1-byte CRC should be sent. The QT1110 returns the expected CRC byte in the same transaction the CRC byte is sent. In this way, the host can immediately determine whether the setup data bytes were received correctly.

Default GUARD_KEY value: 0 (Key 0)
Default GD_EN value: 0 (disabled)
Default CHG value: 0 (data mode)
Default CRC value: 0 (disabled)

7.6 Address 2: Detect Integrator Limit (DIL)/Drift Hold Time (DHT)

Table 7-4. Detect Integrator/Drift Hold Time

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2		DIL				DHT		

DIL: the detection integrator (DI) limit. To suppress false detections caused by spurious events like electrical noise, the device incorporates a DI counter mechanism. A per-key counter is incremented each time the channel has exceeded its threshold and stayed there for a number of acquisitions in succession, without going below the threshold level. When this counter reaches a preset limit the channel is finally declared to be touched. If on any acquisition the delta is not seen to exceed the threshold level, the counter is cleared and the process has to start from the beginning.

Note: A setting of 0 for DI is invalid; the valid range is 1 to 15.

DHT: the drift hold time. After a key-touch has been removed, the QT1110 pauses in the implementation of its "Drift" compensation for a time. After this time has expired, drift compensation continues as normal. The Drift Hold Time is a multiple of 160 ms, providing options from 0 to 2400 ms.

Default DIL value: 3
Default DHT value: 8 (1280 ms)

7.7 Address 3: Positive Threshold (PTHR)/Positive Hysteresis (PHYST)

Table 7-5. Positive Threshold (THR)/Positive Hysteresis (HYST)

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
3		PTHR					PHYST	

PTHR: the positive threshold for the signal. If a key signal is significantly higher than the reference signal, this typically indicates that the calibration data is no longer valid. In other words, some factor has changed since the calibration was carried out, thus rendering it invalid. Generally this is compensated for by the drift, but the greater the difference the longer this will take. In order to speed up this correction, the positive threshold is used: if the positive threshold is exceeded, the QT1110 (that is, all keys) is recalibrated.

PHYST: positive hysteresis. This setting provides a greater degree of control over the implementation of the positive threshold recalibration. The positive hysteresis operates as a "modifier" for the positive threshold. When a key signal is detected as being over the positive threshold, the positive threshold is reduced by a factor corresponding to the positive hysteresis so that the key will not go in and out of positive detection when the signal is on the borderline between drift-compensation of a positive error or recalibration.

The settings for positive hysteresis are:

- 00 = No change to positive threshold
- 01 = 12.5% reduction in positive-detect threshold
- 10 = 25% reduction in positive-detect threshold
- 11 = 37.5% reduction in positive-detect threshold

Default PTHR value: 4 (4 counts above reference)
Default PHYST value: 2 (25% positive hysteresis)



7.8 Address 4: Positive Drift Compensation (PDRIFT)

Table 7-6. Positive Drift Compensation

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
4	0							PDRIFT

When changing ambient conditions cause a change in the key signal, the QT1110 will compensate through its drift functions. *Positive Drift* refers to the case where the signal for a key is greater than the reference.

Drift compensation occurs at a rate of 1 count per drift compensation period.

PDRIFT: the drift compensation period, in multiples of 160 ms. The valid range is 0 to 127, where 0 disables positive drift compensation.

Note: Drift compensation timing is paused while Drift Hold is activated, and continued when Drift Hold has timed out.

Default value: 6 (960 ms)

7.9 Address 5: Positive Recalibration Delay (PRD)

Table 7-7. Positive Recalibration Delay

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
5								PRD

If a key signal is determined to be above the positive threshold, the QT1110 will wait for this delay and confirm that the error condition is still present before initiating a recalibration.

PRD: the positive recalibration delay, in multiples of 160 ms.

Note: All keys are recalibrated in the case of a positive recalibration.

Default value: 6 (960 ms)

7.10 Address 6: Lower Burst Limit (LBL)

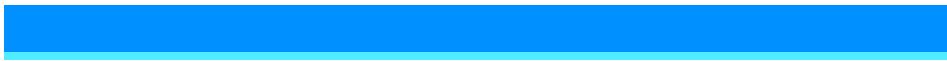
Table 7-8. Lower Burst Limit

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
6								LBL

Normal QTouch signals are in the range of 100 to 1000 counts for each key. The lower burst limit determines the minimum signal that is considered as a valid acquisition. If the count is lower than the lower burst limit, it is considered not to be valid and the key is set to an Error state.

Note: Where a key has a signal of less than the LBL, a detection is not reported on that key.

Default value: 18



7.11 Addresses 7 – 8: AKS Mask

Table 7-9. AKS Mask

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7						AKS_10	AKS_9	AKS_8
8	AKS_7	AKS_6	AKS_5	AKS_4	AKS_3	AKS_2	AKS_1	AKS_0

AKS_n (AKS Mask): 0 = key n AKS disabled, 1 = key n AKS enabled (where n is 0–10).

These bits control which keys have AKS enabled (see [Section 3. on page 9](#)). A “1” means the corresponding key has AKS enabled; a 0 means that the corresponding key has AKS disabled.

Default AKS mask: 0x07 and 0xFF (all keys have AKS enabled)

7.12 Addresses 9 – 15: Detect0 – Detect6 PWM

Each of the 7 detect pins can be configured to output a PWM signal to indicate whether the key is touched (in detect) or not touched (out of detect).

The Detect outputs must be enabled by selecting 7-key mode in the “Device Mode” setting (see [Section 7.4 on page 30](#)), and the corresponding “Key to LED” bits must be set to enable the individual Detect outputs for each key (see [Section 7.14 on page 36](#)).

Table 7-10. Detect0 – Detect6 PWM

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
9						OUT_DETECT0		
10						OUT_DETECT1		
11						OUT_DETECT2		
12						OUT_DETECT3		
13						OUT_DETECT4		
14						OUT_DETECT5		
15						OUT_DETECT6		

IN_DETECTn: PWM to output when key n is “In Detect” (where n is 0–6).

OUT_DETECTn: PWM to output when key n is “Out of Detect” (where n is 0–7). This PWM is also output if the DETECT output is “disconnected” from the key (that is, “LED_n” in address 17 is set to 0), allowing the host to directly control the PWM output.

The values for the “IN_DETECT n ” and “OUT_DETECT n ” nibbles are listed in [Table 7-11](#).

Table 7-11. PWM Values

Value	Meaning
0	0%
1	12.5%
2	25%
3	37.5%
4	50%
5	62.5%
6	75%
7	87.5%
8	100%

Default IN_DETECT n value: 8 (100% PWM – on)
 Default OUT_DETECT n value: 0 (0% PWM – off)

7.13 Address 16: LED Detect Hold Time

Table 7-12. LED Detect Hold Time

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
16	LED_DETECT_HOLD_TIME							

When a key is touched, if the “Detect” outputs and “Key to LED” options are enabled (see [Section 7.12](#) and [Section 7.14](#)), the corresponding “Detect” pin will output its “In-Detect” PWM signal.

After the key touch is removed, the “Detect” output can be held at the “In-Detect” PWM signal for a time before returning to the “Out of Detect” PWM signal. This allows a reasonable length of time for an LED to be illuminated. The length of this time is controlled by the LED Detect Hold Time. Valid values are in multiples of 16 ms.

Default value: 0 (0 ms)



7.14 Address 17: LED Fade/Key to LED

Table 7-13. LED Fade/Key to LED

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
17	FADE	LED_6	LED_5	LED_4	LED_3	LED_2	LED_1	LED_0

FADE: enables/disables fading for all LEDs. This is a global setting; either all LEDs fade, or none of them.

0 = disable (no fade).

1 = enable fading on and off.

LED_n: activates the LED output for the corresponding key output DETECTn (where n is 0–6).

1 = enables the "Detect" output to follow the status of the corresponding key.

0 = disable this function, in which case the "Detect" pin will always output its "Out of Detect" PWM (see [Section 7.12 on page 34](#)).

Default FADE value: 0 (disabled)

Default LED_n value: 1 (enabled)

7.15 Address 18: LED Latch

Table 7-14. LED Latch

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
18	0	LATCH_6	LATCH_5	LATCH_4	LATCH_3	LATCH_2	LATCH_1	LATCH_0

LATCH_n: enables/disables latching of the LED for the corresponding key output DETECTn (where n is 0–6).

1 = enables latching. When latching is enabled for a given LED, the LED toggles its state each time the key is touched.

0 = disables latching.

Note that bit 7 is reserved and should be set to zero.

Default LATCH_n value: 0x00 (latch disabled)

7.16 Addresses 19 – 29: Negative Threshold (NTHR) / Negative Hysteresis (NHYST)

Table 7-15. Negative Threshold / Negative Hysteresis

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
19				KEY_0_NTHR			KEY_0_NHSYT	
20				KEY_1_NTHR			KEY_1_NHSYT	
21				KEY_2_NTHR			KEY_2_NHSYT	
22				KEY_3_NTHR			KEY_3_NHSYT	
23				KEY_4_NTHR			KEY_4_NHSYT	
24				KEY_5_NTHR			KEY_5_NHSYT	
25				KEY_6_NTHR			KEY_6_NHSYT	

Table 7-15. Negative Threshold / Negative Hysteresis (Continued)

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
26					KEY_7_NTHR		KEY_7_NHSYT	
27					KEY_8_NTHR		KEY_8_NHSYT	
28					KEY_9_NTHR		KEY_9_NHSYT	
29					KEY_10_NTHR		KEY_10_NHSYT	

KEY_n_NTHR: the negative threshold for key *n* (where *n* is 0–10).

The negative threshold determines how much the signal must fall (compared to the reference) before a key is considered to be “In Detect”. This level will generally need to be tuned individually for each key. To disable an individual key, set the threshold for that key to 0.

KEY_n_NHYST: the negative hysteresis applied to key *n* detection threshold (where *n* is 0 – 10).

Negative Hysteresis operates as a “modifier” for the negative threshold in order to provide a greater degree of control over the detection of a “Touch”. When a key signal is first detected as being under the negative threshold, the threshold is reduced by a factor corresponding to the selected negative hysteresis. This means that the key will not go in and out of detection when the signal is on the borderline between drift-compensation or touch detection.

The settings for negative hysteresis are:

- 00 No change to negative threshold
- 01 12.5% reduction in negative threshold
- 10 25% reduction in negative threshold
- 11 37.5% reduction in negative threshold

Default KEY_n_NTHR value: 10 counts

Default KEY_n_NHYST value: 2 (25 percent)

7.17 Address 30: Extend Pulse Time

Table 7-16. Extend Pulse Time

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
30				HIGH_TIME			LOW_TIME	

HIGH_TIME: Number of μ s to extend the high pulse time.

LOW_TIME: Number of μ s to extend the low pulse time.

7.18 Addresses 31 – 41: Negative Drift Compensation (NDRIFT) / Negative Recalibration Delay (NRD)

Table 7-17. Negative Drift Compensation / Negative Recalibration Delay

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
31		KEY_0_NDRIFT				KEY_0_NRD		
32		KEY_1_NDRIFT				KEY_1_NRD		
33		KEY_2_NDRIFT				KEY_2_NRD		
34		KEY_3_NDRIFT				KEY_3_NRD		
35		KEY_4_NDRIFT				KEY_4_NRD		
36		KEY_5_NDRIFT				KEY_5_NRD		
37		KEY_6_NDRIFT				KEY_6_NRD		
38		KEY_7_NDRIFT				KEY_7_NRD		
39		KEY_8_NDRIFT				KEY_8_NRD		
40		KEY_9_NDRIFT				KEY_9_NRD		
41		KEY_10_NDRIFT				KEY_10_NRD		

KEY_n_NDRIFT: the negative drift compensation for key n (where n is 0–10).

When changing ambient conditions cause a change in the key signal, the QT1110 will compensate through its drift functions. “Negative Drift” refers to the case where the signal for a key is lower than the reference. Drift compensation occurs at a rate of 1 count per drift compensation period. The entered number is a multiple of 320 ms.

Note that as a key touch, or an approaching touch, naturally causes a negative change in the signal, negative drift should be carried out at a much slower rate than positive drift. Otherwise, a slowly approaching finger will not cause a touch detection, as the falling signal could be compensated through the negative drift mechanism.

Note: Drift compensation timing is paused while Drift Hold is activated, and continues when Drift Hold has timed out.

KEY_n_NRD: the negative recalibration delay for key n (where n is 0 – 10).

In order to avoid a situation where a key remains “stuck” in detect due to, for example, changing environmental conditions, the “Negative Recalibration Delay” sets an upper limit on how long a key can remain “touched”. When this time is exceeded, the QT1110 (that is, all keys) is recalibrated, taking this key (and any others which are in detect) out of detection. This delay is set in a multiple of 2560 ms.

Note: A setting of “0” disables the NRD Timeout.

Default KEY_

n_NDRIFT value: 7 (2240 ms)

Default KEY_

n_NRD value: 10 (25.6 s)

8. Specifications

8.1 Absolute Maximum Specifications

Vdd	-0.5 V to +6 V
Max continuous pin current, any control or drive pin	± 10 mA
Voltage forced onto any pin	-1.0 V to (Vdd + 0.5) V
EEPROM setups maximum writes	100,000 write cycles
	CAUTION: Stresses beyond those listed under <i>Absolute Maximum Specifications</i> may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum specification conditions for extended periods may affect device reliability

8.2 Recommended Operating Conditions

Operating temperature	-40°C to +125°C
Storage temperature	-65°C to +150°C
Vdd	3 V to 5.5 V
Supply ripple + noise	± 20 mV
Cx transverse load capacitance per key	2 pF to 20 pF

8.3 DC Specifications

Vdd = 5.0V, Cs = 4.7 nF, Rs = 1 MΩ, Ta = recommended range, unless otherwise noted

Parameter	Description	Min	Typ	Max	Units	Notes
Iddr	Average supply current, running	–	–	12 at 5 V 8 at 3 V	mA	For typical values see Section 8.8
Vil	Low input logic level	-0.5V	–	0.3 × Vdd	V	
Vih	High input logic level	0.6 × Vdd	Vdd	Vdd + 0.5	V	
Vol	Low output voltage	0	–	0.7	V	10 mA sink current
Voh	High output voltage	0.8 × Vdd	–	Vdd	V	10 mA source current
Iil	Input leakage current	–	<0.05	1	µA	
Rrst	Internal RST pull-up resistor	30	–	60	kΩ	



8.4 Timing Specifications

Parameter	Description	Min	Typ	Max	Units	Notes
T _{BS}	Burst duration	–	5	–	ms	4.7 nF Cs
F _C	Burst center frequency	–	53	–	kHz	
F _M	Burst modulation, percentage	–	18	–	%	
T _{PW}	Pulse width	–	6000	–	ns	

8.5 SPI Bus Specifications

8.5.1 General Specifications

Parameter	Specification
Address space	8-bit
Maximum clock rate	1.5 MHz
Minimum low clock period	333 ns
Minimum high clock period	333 ns
Clock idle	High
Setup on	Leading (falling) edge
Clock out on	Trailing (rising) edge
SPI Enable delay (SS low to SCK low)	1 µs minimum

8.5.2 Full SPI Mode

Parameter	Specification
Minimum time between bytes	150 µs
Minimum time between communications	Generally 150 µs; longer delays required to implement some commands, as follows: <ul style="list-style-type: none"> • Send Setups: 150 µs after all setup bytes are returned • Calibrate All: 150 µs • Calibrate Key: 150 µs • Reset: 160 ms • Sleep: 150 µs after a low signal is applied to SS or CHANGE to wake the device • Store to EEPROM: 200 ms • Restore from EEPROM: 150 ms • Erase EEPROM: 50 ms • Recover EEPROM: 50 ms

8.5.3 Quick SPI Mode

Parameter	Specification
Minimum time between bytes	50 µs
Minimum time between communications	Generally 50 µs, except for the following: <ul style="list-style-type: none"> • Store to EEPROM: 200 ms • Switch to Full SPI: 150 µs

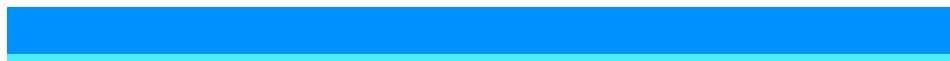
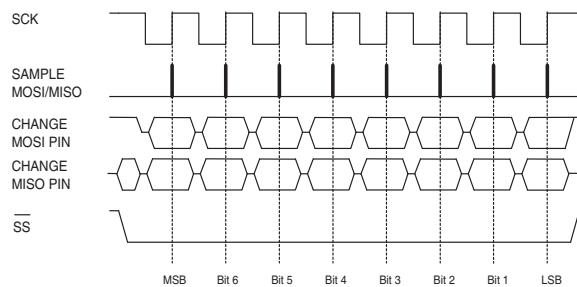


Figure 8-1. Data Byte Exchange



8.6 External Reset

Parameter	Description	Operation
V_{RST}	Threshold voltage low (Activate) Threshold voltage high (Release)	$0.2 \times Vdd$ $0.9 \times Vdd$
Reset	Minimum length of Reset low	600 ns at 5 V 1100 ns at 3 V

8.7 Internal Resonator

Parameter	Operation
Internal RC oscillator	8 MHz with spread-spectrum modifier during measurement bursts

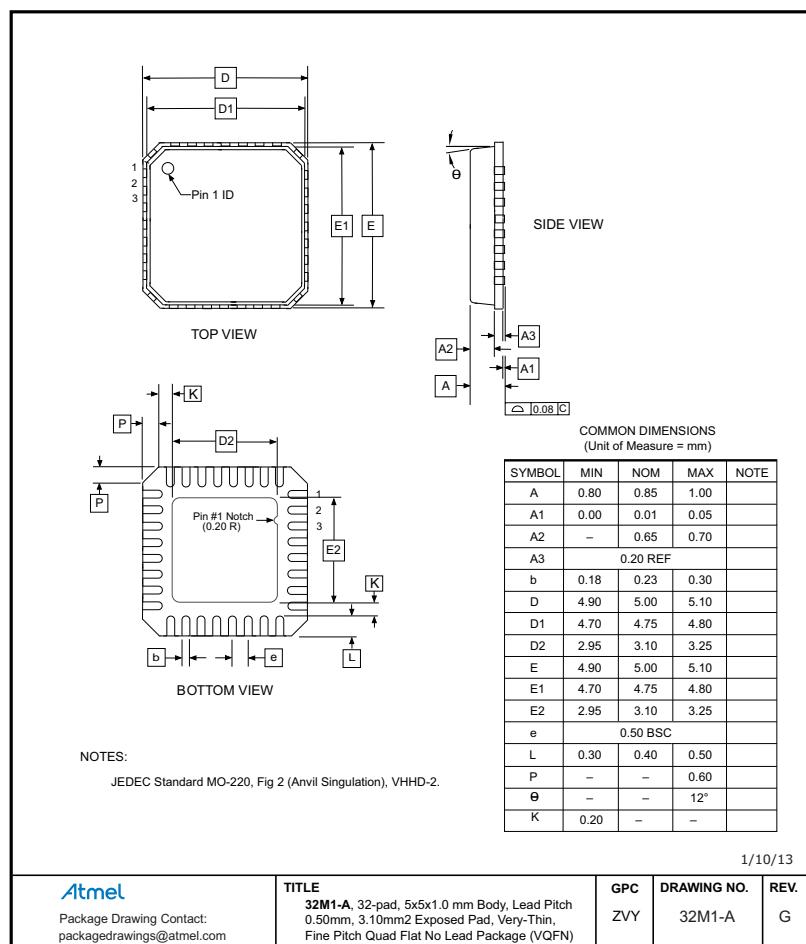
8.8 Power Consumption

		7-key Parallel		7-key Serial		11-key Parallel		11-key Serial		11-key Serial, 1 key enabled	
Vdd (V)	Cycle Time	Actual Cycle Time (ms)	Idd (µA)	Actual Cycle Time (ms)	Idd (µA)						
4.7 nF Cs Capacitors											
3.0	0 (Free Run)	13.2	2470	26.6	2350	15.3	2385	37.4	2420	2.15	2107
	1 (16 ms Nominal)	17.2	2180	26.6	2350	17.3	2182	37.4	2420	16.5	950
	2 (32 ms Nominal)	33.6	1470	34.4	1950	33.8	1435	37.4	2420	33	739
	4 (64 ms Nominal)	66.4	1010	67.2	1325	66.4	1045	68.4	1587	66	691
	8 (128 ms Nominal)	132	840	133	1025	132	840	134	1120	132	668
	15 (240 ms Nominal)	248	815	250	850	250	810	250	1008	248	656
5.0	0 (Free Run)	15.1	5530	30.2	5405	17.3	5674	43.6	5425	2.15	4860
	1 (16 ms Nominal)	17.2	5290	30.2	5405	17.3	5674	43.6	5425	16.3	2965
	2 (32 ms Nominal)	33.4	4210	34.4	5350	33.6	4013	43.6	5425	32.6	2400
	4 (64 ms Nominal)	65.6	3120	66.8	4015	65.6	3240	67.6	4130	64.8	2248
	8 (128 ms Nominal)	130	2705	132	3225	130	2840	132	3530	129	2206
	15 (240 ms Nominal)	244	2440	244	3035	246	2465	245	3015	244	2163
10 nF Cs Capacitors											
3.0	0 (Free Run)	24.2	2375	48.4	2430	24.2	2434	63.6	2416	8.6	2130
	1 (16 ms Nominal)	24.2	2375	48.4	2430	24.2	2434	63.6	2416	16.7	1422
	2 (32 ms Nominal)	34.4	1860	48.4	2430	34	1945	63.6	2416	33	1065
	4 (64 ms Nominal)	66.8	1285	68.4	1910	66.4	1290	69.6	2260	65	848
	8 (128 ms Nominal)	131	995	133	1320	132	980	134	1485	130	766
	15 (240 ms Nominal)	246	845	248	1030	246	824	248	1080	243	708
5.0	0 (Free Run)	26	5810	56.4	5510	28	5675	73.6	5596	8.6	5145
	1 (16 ms Nominal)	26	5810	56.4	5510	28	5675	73.6	5596	16.6	3990
	2 (32 ms Nominal)	34	5170	56.4	5510	34	5196	73.6	5596	32.6	3160
	4 (64 ms Nominal)	66	3990	67.6	5120	66.4	3780	73.6	5596	64.8	2690
	8 (128 ms Nominal)	131	3290	132	3850	130	2910	133	4055	129	2310
	15 (240 ms Nominal)	244	2950	244	3310	242	2675	246	3170	241	2270

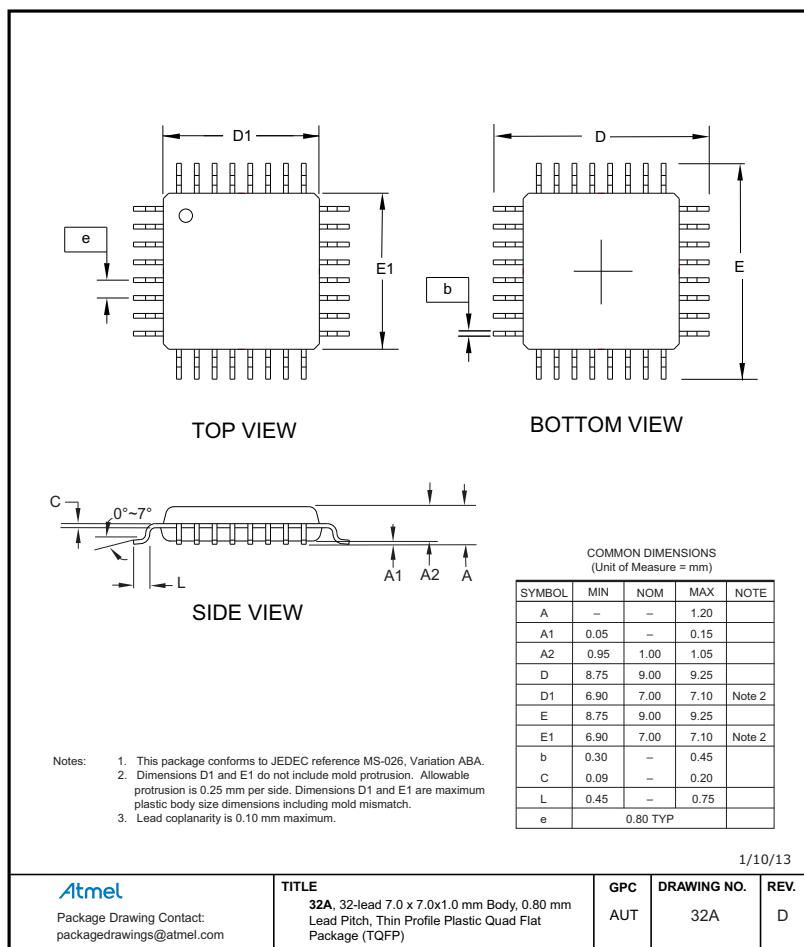
Note: These values are for reference only; values are untested.

8.9 Mechanical Dimensions

8.9.1 AT42QT1110-MZ – 32-pin 5 x 5 mm QFN

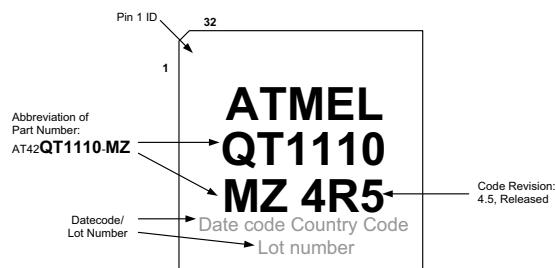


8.9.2 AT42QT1110-AZ – 32-pin 7 x 7 mm TQFP

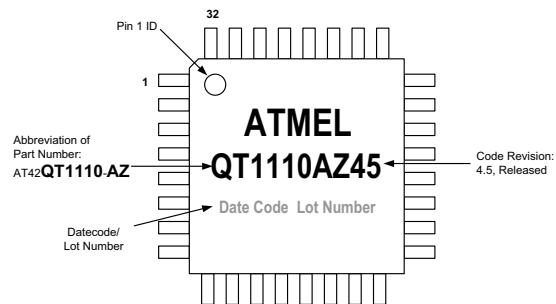


8.10 Marking

8.10.1 AT42QT1110-MZ – 32-pin 5 x 5 mm QFN



8.10.2 AT42QT1110-AZ – 32-pin 7 x 7 mm TQFP





8.11 Part Number

Part Number	Description
AT42QT1110-MZ	32-pin 5 x 5 mm QFN RoHS compliant (-40°C to +125°C)
AT42QT1110-AZ	32-pin 7 x 7 mm TQFP RoHS compliant (-40°C to +125°C)

8.12 Moisture Sensitivity Level (MSL)

MSL Rating	Peak Body Temperature	Specifications
MSL3	260°C	IPC/JEDEC J-STD-020

Appendix A. CRC Calculation

If the use of a cyclic redundancy check (CRC) during data transmission is enabled, the host must generate a valid CRC so that this can be correctly compared to the corresponding CRC generated by the QT1110. This appendix gives example C code to show how the CRC can be generated by the host.

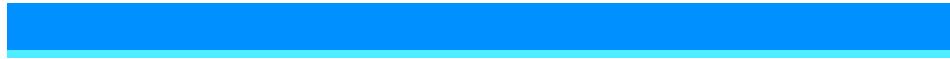
```
/*=====
unsigned char calc_crc(unsigned char crc, unsigned char data)
-----
Purpose: Calculate CRC for data packets
Input : CRC, Data
Output : Updated CRC
Notes : -
=====
unsigned char calc_crc(unsigned char crc, unsigned char data)
{
    unsigned char index;
    unsigned char fb;
    index = 8;
    do
    {
        fb = (crc ^ data) & 0x01u;
        data >>= 1u;
        crc >>= 1u;
        if(fb)
        {
            crc ^= 0x8c;
        }
    } while(--index);
    return crc;
}

/* Example Calling Routine */
unsigned char calculate_config_checksum(void)
{
    int i;
    unsigned char CRC_val = 0;
    unsigned char setup_data[42] =
    {
        0xB2, 0x00, 0x38, 0x12, 0x06, 0x06, 0x12, 0x07, 0xFF, 0x80,
        0x80, 0x80, 0x80, 0x80, 0x80, 0x32, 0xEF, 0x00, 0x29,
        0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
        0x00, 0x7A, 0x7A, 0x7A, 0x7A, 0x7A, 0x7A, 0x7A, 0x7A,
        0x7A, 0x7A
    };

    for(i = 0; i < sizeof(setup_data); i++)
    {
        CRC_val = calc_crc(CRC_val, setup_data[i]);
    }
    return(CRC_val);
}
```

Revision History

Revision No.	History
Revision A – November 2008	<ul style="list-style-type: none">Initial Release
Revision B – December 2008	<ul style="list-style-type: none">Updated for chip revision 2.1
Revision C – December 2008	<ul style="list-style-type: none">Updated SPI specifications
Revision D – February 2009	<ul style="list-style-type: none">Updated for chip revision 3.1
Revision E – April 2009	<ul style="list-style-type: none">Updated for chip revision 3.2: added self-test function
Revision F – July 2009	<ul style="list-style-type: none">Updated for chip revision 4.3: added Quick SPI mode
Revision G – October 2009	<ul style="list-style-type: none">Updated specificationsErratum note added concerning CRC calculations for chip revision 4.3
Revision H – February 2010	<ul style="list-style-type: none">Updated for chip revision 4.4CRC calculations updated
Revision I – March 2010	<ul style="list-style-type: none">Updated for chip revision 4.5
Revision J – March 2013	<ul style="list-style-type: none">General updatesApply new template

**Atmel Corporation**

1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1) (408) 441-0311
Fax: (+1) (408) 487-2600
www.atmel.com

Atmel Asia Limited

Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel München GmbH

Business Campus
Parking 4
D-85748 Garching bei München
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.

16F Shin-Osaki Kangyo Bldg
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81) (3) 6417-0300
Fax: (+81) (3) 6417-0370

© 2013 Atmel Corporation. All rights reserved. / Rev.: 9570J-AT42-05/2013

Atmel®, Atmel logo and combinations thereof, QTouch®, Adjacent Key Suppression®, AKS®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be registered trademarks or trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATTEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATTEL WEBSITE, ATTEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. ATTEL SHALL BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, SPECIAL, OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT; EVEN IF ATTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

D.4 ENDSTOP DATASHEET

D2FS

Ultra Subminiature Basic Switch

Simple construction and high reliability for long time

- Developed for a few number of operations during a long period of use such as door-detection to avoid sabotage on meters.
- Simple construction with a single-leaf movable spring realizes reasonable cost.
- Self-clinching, right-angled and left-angled PCB terminals are also available.

RoHS Compliant



NEW

Model Number Legend

D2FS-F□-N□

- | | |
|--|--|
| 1. Actuator
None : Pin plunger
L : Hinge lever | 2. Terminals
None : PCB terminals (Straight)
-T : PCB terminals (Self-clinching)
-A : PCB terminals (Right-angled)
-A1 : PCB terminals (Left-angled) |
|--|--|

List of Models

Actuator	Terminals	Model
	PCB terminals (Straight)	D2FS-F-N
	PCB terminals (Self-clinching)	D2FS-F-N-T
	PCB terminals (Right-angled)	D2FS-F-N-A
	PCB terminals (Left-angled)	D2FS-F-N-A1
	PCB terminals (Straight)	D2FS-FL-N
	PCB terminals (Self-clinching)	D2FS-FL-N-T
	PCB terminals (Right-angled)	D2FS-FL-N-A
	PCB terminals (Left-angled)	D2FS-FL-N-A1

Contact Form

- SPST-NO



Contact Specifications

Contact	Specification	Crossbar
	Material	Silver
	Gap (standard value)	0.4 mm
	Minimum applicable load (see note)	1 mA at 5 VDC

Note: For more information on the minimum applicable load, refer to *Using Micro Loads*.

D2FS**Ultra Subminiature Basic Switch****Ratings**

Rated voltage	Resistive load
6 VDC	0.1 A

Note: The rating values apply under the following test conditions.

Ambient temperature: 20 ± 2°C

Ambient humidity: 65 ± 5%

Operating frequency: 20 operations/min

Characteristics

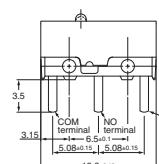
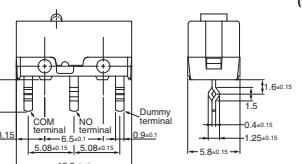
Operating speed	Pin plunger models: 1 mm to 500 mm/s Lever models: 5 mm to 500 mm/s
Operating frequency	Pin plunger models: 200 operations/min max. Lever models: 100 operations/min max.
Insulation resistance	30 operations/min max.
Contact resistance (initial value)	100 MΩ min. (at 500 VDC)
	100 mΩ max.
Dielectric strength	Between terminals of same polarity 600 VAC 50/60 Hz 1 min
	Between current carrying metal parts and ground 1,500 VAC 50/60 Hz 1 min
Vibration resistance	Between each terminal and non-current carrying metal part
Shock resistance	Malfunction *1 10 to 55 Hz, 1.5 mm double amplitude
Durability *2	Destruction 1,000 m/s ² max.
	Malfunction *1 300 m/s ² max.
Degree of protection	100,000 operations min. (at 30 ops./min.)
Ambient operating temperature	100,000 operations min. (at 30 ops./min.)
Ambient operation humidity	IP40
Weight	-20 to +70°C (at 60%RH Max.) (with no icing or condensation)
	85%RH max. (for +5 to +35°C)
	Approx. 0.5 g (pin plunger models)

Note: The data given above are initial values.

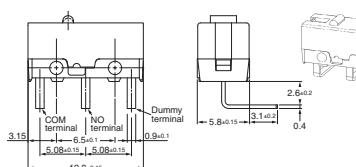
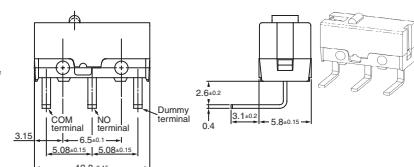
*1. The values are at Free Position and Total Travel Position values for pin plunger, and Total Travel Position value for lever.

Close or open circuit of the contact is 1 ms max.

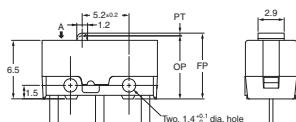
*2. For testing conditions, consult your OMRON sales representative.

D2FS**Ultra Subminiature Basic Switch****Terminals (Unit: mm)****● PCB Terminals (Straight)****● PCB terminals (Self-clinching)****PCB Mounting Dimensions (Reference)**

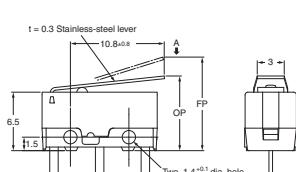
Don't connect the dummy terminal to the circuit.

● PCB Terminals (Right-angled)**● PCB Terminals (Left-angled)****Dimensions (Unit: mm) / Operating Characteristics**

The following illustrations and drawings are for D2FS models with PCB terminals (straight). Self-clinching, and right-angled, left angled terminals are omitted from the following drawings. Refer to the above for these terminals.
When ordering, replace □ with the code for the terminal that you need. See the "List of Models" for available combinations of models.

● Pin Plunger Models**D2FS-F-N□**

Model	D2FS-F-N□
OF max.	0.88 N (87 gf)
RF min.	0.05 N (5 gf)
PT max.	0.5 mm
OT min.	0.25 mm
MD max.	0.12 mm
FP max.	7.5 mm
OP	7.0 ± 0.3 mm

● Hinge Lever Models**D2FS-FL-N□**

Model	D2FS-FL-N□
OF max.	0.25 N (25 gf)
RF min.	0.02 N (2 gf)
OT min.	0.55 mm
MD max.	0.5 mm
FP max.	11.5 mm
OP	8.3 ± 1.2 mm

Note: 1. Unless otherwise specified, a tolerance of ±0.4 mm applies to all dimensions.
2. The operating characteristics are for operation in the A direction (↓).

D2FS

Ultra Subminiature Basic Switch

Precautions

*Refer to General Information.

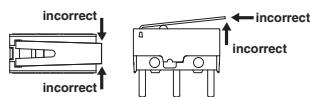
Cautions	Correct Use
----------	-------------

● Soldering

- When using automatic soldering baths, we recommend soldering at 260°C within 5 seconds. Make sure that the liquid surface of the solder does not flow over the edge of the board.
- When soldering terminals manually, perform soldering within 3 seconds at iron tip temperature not higher than 350°C. Do not apply any external force for at least 1 minute after soldering. When applying solder, keep the solder away from the case of the Switch and do not allow solder or flux to flow into the case.

● Side-actuated (Cam/Dog) Operation

- When using a cam or dog to operate the Switch, factors such as the operating speed, operating frequency, push-button indentation, and material and shape of the cam or dog will affect the durability of the Switch. Confirm performance specifications under actual operating conditions before using the Switch in applications.



● Using Micro Loads

- Even when using micro load models within the operating range, inrush currents or surges may decrease the life expectancy of the Switch. Therefore, insert a contact protection circuit where necessary.

● Application Environment

- Do not use the Switch in locations that are subject to toxic gas, silicon, excessive dust, excessive dirt, high temperatures, high humidity, sudden temperature changes, water splashes, or oil splashes. Otherwise, damage resulting by faulty contact of the Switch contacts, corrosion, or other causes, or other functional faults may occur.

• Application examples provided in this document are for reference only. In actual applications, confirm equipment functions and safety before using the product.
 • Consult your OMRON representative before using the product under conditions which are not described in the manual or applying the product to nuclear control systems, railroad systems, aviation systems, vehicles, combustion systems, medical equipment, amusement machines, safety equipment, and other systems or equipment that may have a serious influence on lives and property if used improperly. Make sure that the ratings and performance characteristics of the product provide a margin of safety for the system or equipment, and be sure to provide the system or equipment with double safety mechanisms.

Note: Do not use this document to operate the Unit.

OMRON Corporation
 Electronic and Mechanical Components Company

Contact: www.omron.com/ecb

Cat. No. B121-E1-04
 0117(0207)(O)

D.5 OPERATIONAL AMPLIFIER DATASHEET


MCP601/1R/2/3/4
2.7V to 6.0V Single Supply CMOS Op Amps
Features

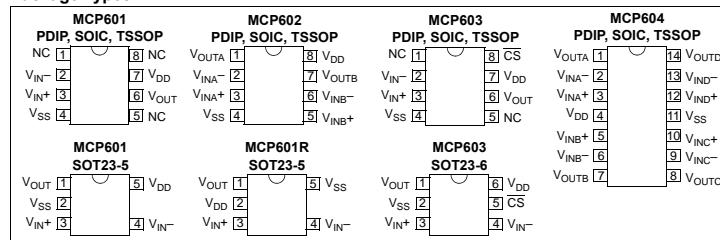
- Single-Supply: 2.7V to 6.0V
- Rail-to-Rail Output
- Input Range Includes Ground
- Gain Bandwidth Product: 2.8 MHz (typical)
- Unity-Gain Stable
- Low Quiescent Current: 230 μ A/amplifier (typical)
- Chip Select (CS): **MCP603 only**
- Temperature Ranges:
 - Industrial: -40°C to +85°C
 - Extended: -40°C to +125°C
- Available in Single, Dual, and Quad

Typical Applications

- Portable Equipment
- A/D Converter Driver
- Photo Diode Pre-amp
- Analog Filters
- Data Acquisition
- Notebooks and PDAs
- Sensor Interface

Available Tools

- SPICE Macro Models
- FilterLab® Software
- Mindi™ Simulation Tool
- MAPS (Microchip Advanced Part Selector)
- Analog Demonstration and Evaluation Boards
- Application Notes

Package Types

MCP601/1R/2/3/4

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings †

$V_{DD} - V_{SS}$	7.0V				
Current at Input Pins	±2 mA				
Analog Inputs (V_{IN^+}, V_{IN^-}) ‡	$V_{SS} - 1.0V$ to $V_{DD} + 1.0V$				
All Other Inputs and Outputs	$V_{SS} - 0.3V$ to $V_{DD} + 0.3V$				
Difference Input Voltage	$ V_{DD} - V_{SS} $				
Output Short Circuit Current	Continuous				
Current at Output and Supply Pins	±30 mA				
Storage Temperature	-65°C to +150°C				
Maximum Junction Temperature (T_J)	+150°C				
ESD Protection On All Pins (HBM; MM)	≥ 3 kV; 200V				

† Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

‡ See Section 4.1.2 "Input Voltage and Current Limits".

DC CHARACTERISTICS

Electrical Specifications: Unless otherwise specified, $T_A = +25^\circ C$, $V_{DD} = +2.7V$ to $+5.5V$, $V_{SS} = GND$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, and $R_L = 100\text{ k}\Omega$ to V_L , and C_S is tied low. (Refer to Figure 1-2 and Figure 1-3).

Parameters	Sym	Min	Typ	Max	Units	Conditions
Input Offset						
Input Offset Voltage	V_{OS}	-2	±0.7	+2	mV	
Industrial Temperature	V_{OS}	-3	±1	+3	mV	$T_A = -40^\circ C$ to $+85^\circ C$ (Note 1)
Extended Temperature	V_{OS}	-4.5	±1	+4.5	mV	$T_A = +40^\circ C$ to $+125^\circ C$ (Note 1)
Input Offset Temperature Drift	$\Delta V_{OS}/\Delta T_A$	—	±2.5	—	µV/°C	$T_A = -40^\circ C$ to $+125^\circ C$
Power Supply Rejection	PSRR	80	88	—	dB	$V_{DD} = 2.7V$ to $5.5V$
Input Current and Impedance						
Input Bias Current	I_B	—	1	—	pA	
Industrial Temperature	I_B	—	20	60	pA	$T_A = +85^\circ C$ (Note 1)
Extended Temperature	I_B	—	450	5000	pA	$T_A = +125^\circ C$ (Note 1)
Input Offset Current	I_{OS}	—	±1	—	pA	
Common Mode Input Impedance	Z_{CM}	—	$10^{13} 6$	—	Ω pF	
Differential Input Impedance	Z_{DIFF}	—	$10^{13} 3$	—	Ω pF	
Common Mode						
Common Mode Input Range	V_{CMR}	$V_{SS} - 0.3$	—	$V_{DD} - 1.2$	V	
Common Mode Rejection Ratio	CMRR	75	90	—	dB	$V_{DD} = 5.0V$, $V_{CM} = -0.3V$ to $3.8V$
Open-loop Gain						
DC Open-loop Gain (large signal)	A_{OL}	100	115	—	dB	$R_L = 25\text{ k}\Omega$ to V_L , $V_{OUT} = 0.1V$ to $V_{DD} - 0.1V$
	A_{OL}	95	110	—	dB	$R_L = 5\text{ k}\Omega$ to V_L , $V_{OUT} = 0.1V$ to $V_{DD} - 0.1V$
Output						
Maximum Output Voltage Swing	V_{OL}, V_{OH}	$V_{SS} + 15$	—	$V_{DD} - 20$	mV	$R_L = 25\text{ k}\Omega$ to V_L , Output overdrive = 0.5V
	V_{OL}, V_{OH}	$V_{SS} + 45$	—	$V_{DD} - 60$	mV	$R_L = 5\text{ k}\Omega$ to V_L , Output overdrive = 0.5V
Linear Output Voltage Swing	V_{OUT}	$V_{SS} + 100$	—	$V_{DD} - 100$	mV	$R_L = 25\text{ k}\Omega$ to V_L , $A_{OL} \geq 100\text{ dB}$
	V_{OUT}	$V_{SS} + 100$	—	$V_{DD} - 100$	mV	$R_L = 5\text{ k}\Omega$ to V_L , $A_{OL} \geq 95\text{ dB}$
Output Short Circuit Current	I_{SC}	—	±22	—	mA	$V_{DD} = 5.5V$
	I_{SC}	—	±12	—	mA	$V_{DD} = 2.7V$
Power Supply						
Supply Voltage	V_{DD}	2.7	—	6.0	V	(Note 2)
Quiescent Current per Amplifier	I_Q	—	230	325	µA	$I_Q = 0$

Note 1: These specifications are not tested in either the SOT-23 or TSSOP packages with date codes older than YYWW = 0408. In these cases, the minimum and maximum values are by design and characterization only.

2: All parts with date codes November 2007 and later have been screened to ensure operation at $V_{DD}=6.0V$. However, the other minimum and maximum specifications are measured at 1.4V and/or 5.5V.

MCP601/1R/2/3/4**AC CHARACTERISTICS**

Electrical Specifications: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{ V}$ to $+5.5\text{ V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, and $R_L = 100\text{ k}\Omega$ to V_L , $C_L = 50\text{ pF}$, and CS is tied low. (Refer to Figure 1-2 and Figure 1-3).

Parameters	Sym	Min	Typ	Max	Units	Conditions
Frequency Response						
Gain Bandwidth Product	GBWP	—	2.8	—	MHz	
Phase Margin	PM	—	50	—	°	$G = +1\text{ V/V}$
Step Response						
Slew Rate	SR	—	2.3	—	V/ μs	$G = +1\text{ V/V}$
Settling Time (0.01%)	t_{settle}	—	4.5	—	μs	$G = +1\text{ V/V}$, 3.8V step
Noise						
Input Noise Voltage	E_{ni}	—	7	—	$\mu\text{V}_{P,P}$	$f = 0.1\text{ Hz}$ to 10 Hz
Input Noise Voltage Density	e_{ni}	—	29	—	$\text{nV}/\sqrt{\text{Hz}}$	$f = 1\text{ kHz}$
	e_{ni}	—	21	—	$\text{nV}/\sqrt{\text{Hz}}$	$f = 10\text{ kHz}$
Input Noise Current Density	i_{ni}	—	0.6	—	$\text{fA}/\sqrt{\text{Hz}}$	$f = 1\text{ kHz}$

MCP603 CHIP SELECT (CS) CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{ V}$ to $+5.5\text{ V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, and $R_L = 100\text{ k}\Omega$ to V_L , $C_L = 50\text{ pF}$, and CS is tied low. (Refer to Figure 1-2 and Figure 1-3).

Parameters	Sym	Min	Typ	Max	Units	Conditions
CS Low Specifications						
CS Logic Threshold, Low	V_{IL}	V_{SS}	—	0.2 V_{DD}	V	
CS Input Current, Low	I_{CSL}	-1.0	—	—	μA	$\overline{\text{CS}} = V_{DD}/2$
CS High Specifications						
CS Logic Threshold, High	V_{IH}	0.8 V_{DD}	—	V_{DD}	V	
CS Input Current, High	I_{CSH}	—	0.7	2.0	μA	$\overline{\text{CS}} = V_{DD}$
Shutdown V_{SS} current	I_{O_SHDN}	-2.0	-0.7	—	μA	$\overline{\text{CS}} = V_{DD}$
Amplifier Output Leakage in Shutdown	I_{O_SHDN}	—	1	—	nA	
Timing						
$\overline{\text{CS}}$ Low to Amplifier Output Turn-on Time	t_{ON}	—	3.1	10	μs	$\overline{\text{CS}} \leq 0.2V_{DD}$, $G = +1\text{ V/V}$
$\overline{\text{CS}}$ High to Amplifier Output High-Z Time	t_{OFF}	—	100	—	ns	$\overline{\text{CS}} \geq 0.8V_{DD}$, $G = +1\text{ V/V}$, No load.
Hysteresis	V_{HYST}	—	0.4	—	V	$V_{DD} = 5.0\text{ V}$

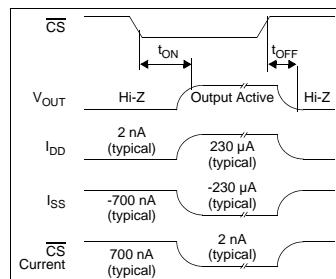


FIGURE 1-1: MCP603 Chip Select ($\overline{\text{CS}}$) Timing Diagram

MCP601/1R/2/3/4

TEMPERATURE CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, $V_{DD} = +2.7V$ to $+5.5V$ and $V_{SS} = GND$.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Temperature Ranges						
Specified Temperature Range	T_A	-40	—	+85	°C	Industrial temperature parts
	T_A	-40	—	+125	°C	Extended temperature parts
Operating Temperature Range	T_A	-40	—	+125	°C	Note
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistances						
Thermal Resistance, 5L-SOT23	θ_{JA}	—	256	—	°C/W	
Thermal Resistance, 6L-SOT23	θ_{JA}	—	230	—	°C/W	
Thermal Resistance, 8L-PDIP	θ_{JA}	—	85	—	°C/W	
Thermal Resistance, 8L-SOIC	θ_{JA}	—	163	—	°C/W	
Thermal Resistance, 8L-TSSOP	θ_{JA}	—	124	—	°C/W	
Thermal Resistance, 14L-PDIP	θ_{JA}	—	70	—	°C/W	
Thermal Resistance, 14L-SOIC	θ_{JA}	—	120	—	°C/W	
Thermal Resistance, 14L-TSSOP	θ_{JA}	—	100	—	°C/W	

Note: The Industrial temperature parts operate over this extended range, but with reduced performance. The Extended temperature specs do not apply to Industrial temperature parts. In any case, the internal Junction temperature (T_J) must not exceed the absolute maximum specification of 150°C.

1.1 Test Circuits

The test circuits used for the DC and AC tests are shown in [Figure 1-2](#) and [Figure 1-3](#). The bypass capacitors are laid out according to the rules discussed in [Section 4.5 "Supply Bypass"](#).

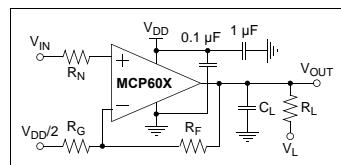


FIGURE 1-2: AC and DC Test Circuit for Most Non-Inverting Gain Conditions.

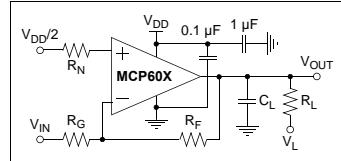


FIGURE 1-3: AC and DC Test Circuit for Most Inverting Gain Conditions.

MCP601/1R/2/3/4**2.0 TYPICAL PERFORMANCE CURVES**

Note: The graphs and tables provided following this note are a statistical summary based on a limited number of samples and are provided for informational purposes only. The performance characteristics listed herein are not tested or guaranteed. In some graphs or tables, the data presented may be outside the specified operating range (e.g., outside specified power supply range) and therefore outside the warranted range.

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100 \text{ k}\Omega$ to V_L , $C_L = 50 \text{ pF}$ and CS is tied low.

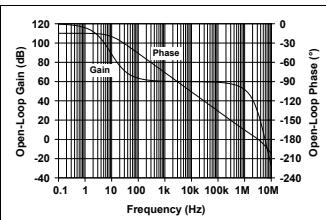


FIGURE 2-1: Open-Loop Gain, Phase vs. Frequency.

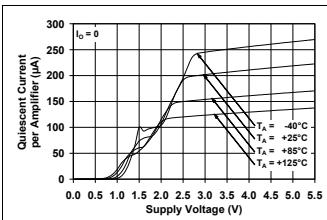


FIGURE 2-4: Quiescent Current vs. Supply Voltage.

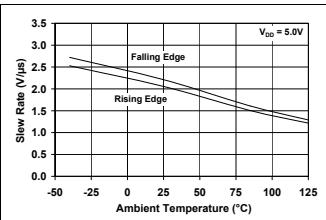


FIGURE 2-2: Slew Rate vs. Temperature.

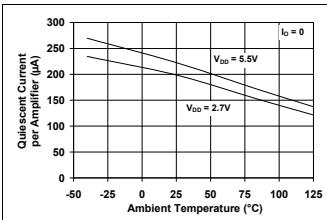


FIGURE 2-5: Quiescent Current vs. Temperature.

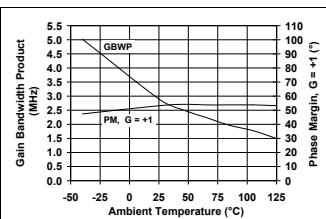


FIGURE 2-3: Gain Bandwidth Product, Phase Margin vs. Temperature.

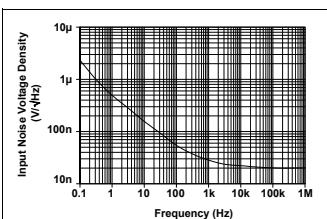


FIGURE 2-6: Input Noise Voltage Density vs. Frequency.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100\text{k}\Omega$ to V_L , $C_L = 50\text{pF}$ and CS is tied low.

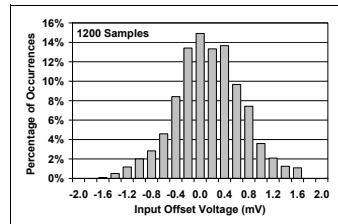


FIGURE 2-7: Input Offset Voltage.

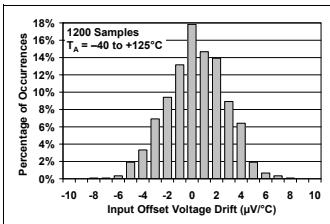


FIGURE 2-10: Input Offset Voltage Drift.

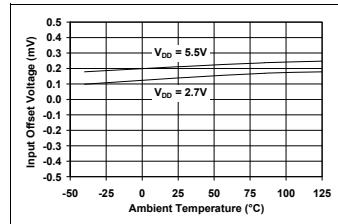


FIGURE 2-8: Input Offset Voltage vs. Temperature.

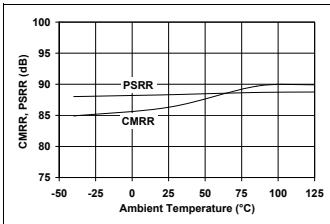


FIGURE 2-11: CMRR, PSRR vs. Temperature.

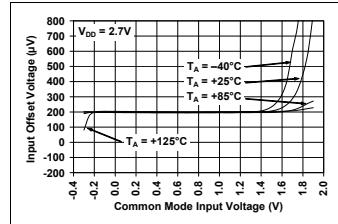


FIGURE 2-9: Input Offset Voltage vs. Common Mode Input Voltage with $V_{DD} = 2.7\text{V}$.

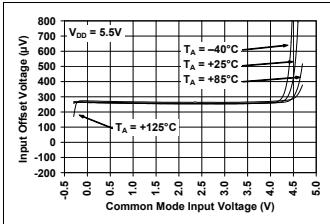


FIGURE 2-12: Input Offset Voltage vs. Common Mode Input Voltage with $V_{DD} = 5.5\text{V}$.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100\text{k}\Omega$ to V_L , $C_L = 50\text{pF}$ and CS is tied low.

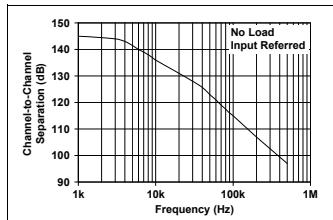


FIGURE 2-13: Channel-to-Channel Separation vs. Frequency.

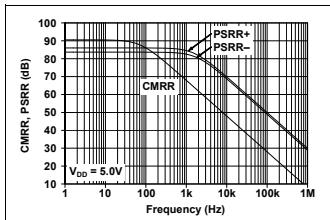


FIGURE 2-16: CMRR, PSRR vs. Frequency.

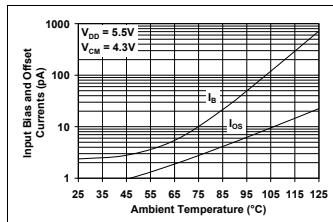


FIGURE 2-14: Input Bias Current, Input Offset Current vs. Ambient Temperature.

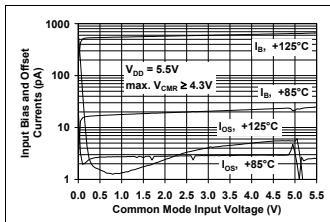


FIGURE 2-17: Input Bias Current, Input Offset Current vs. Common Mode Input Voltage.

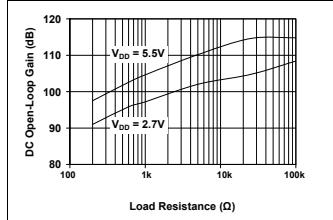


FIGURE 2-15: DC Open-Loop Gain vs. Load Resistance.

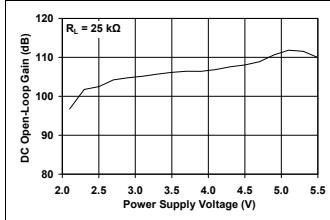


FIGURE 2-18: DC Open-Loop Gain vs. Supply Voltage.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100\text{k}\Omega$ to V_L , $C_L = 50\text{pF}$ and CS is tied low.

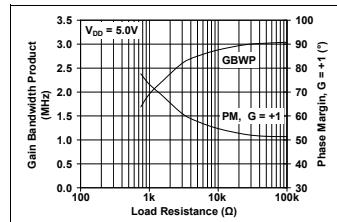


FIGURE 2-19: Gain Bandwidth Product, Phase Margin vs. Load Resistance.

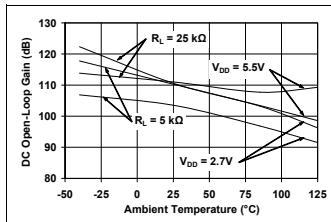


FIGURE 2-22: DC Open-Loop Gain vs. Temperature.

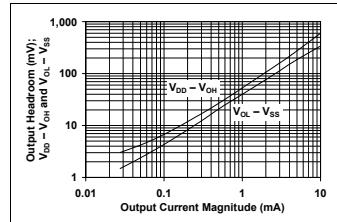


FIGURE 2-20: Output Voltage Headroom vs. Output Current.

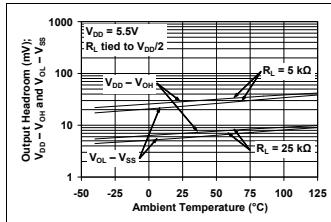


FIGURE 2-23: Output Voltage Headroom vs. Temperature.

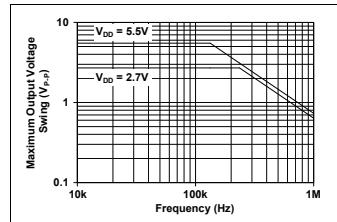


FIGURE 2-21: Maximum Output Voltage Swing vs. Frequency.

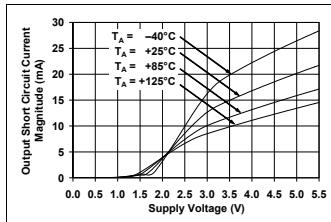


FIGURE 2-24: Output Short-Circuit Current vs. Supply Voltage.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100 \text{k}\Omega$ to V_L , $C_L = 50 \text{pF}$ and CS is tied low.

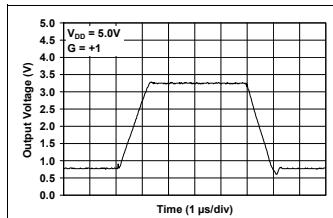


FIGURE 2-25: Large Signal Non-Inverting Pulse Response.

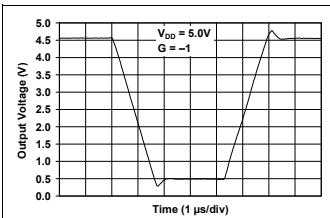


FIGURE 2-28: Large Signal Inverting Pulse Response.

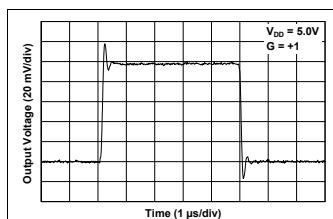


FIGURE 2-26: Small Signal Non-Inverting Pulse Response.

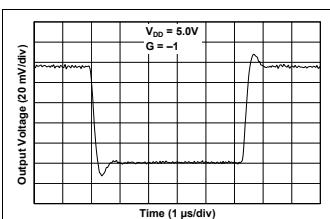


FIGURE 2-29: Small Signal Inverting Pulse Response.

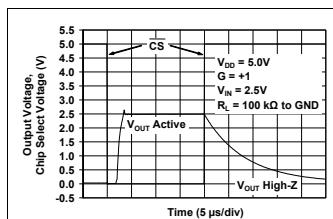


FIGURE 2-27: Chip Select Timing (MCP603).

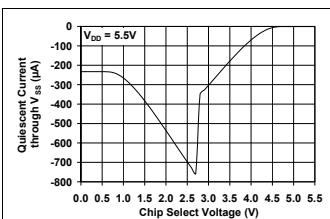


FIGURE 2-30: Quiescent Current Through V_{SS} vs. Chip Select Voltage (MCP603).

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100\text{k}\Omega$ to V_L , $C_L = 50\text{pF}$ and CS is tied low.

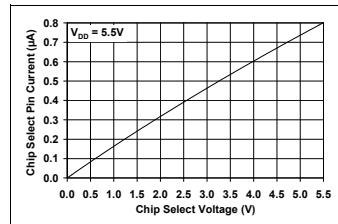


FIGURE 2-31: Chip Select Pin Input Current vs. Chip Select Voltage.

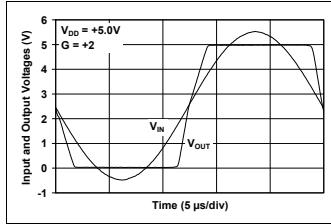


FIGURE 2-33: The MCP601/1R/2/3/4 family of op amps shows no phase reversal under input overdrive.

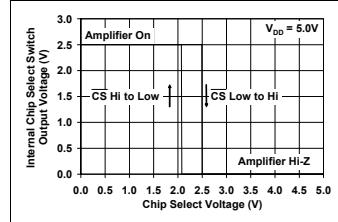


FIGURE 2-32: Hysteresis of Chip Select's Internal Switch.

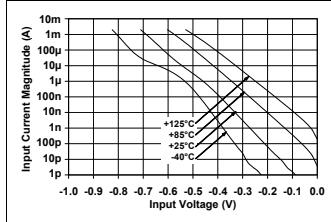


FIGURE 2-34: Measured Input Current vs. Input Voltage (below V_{SS}).

MCP601/1R/2/3/4**3.0 PIN DESCRIPTIONS**

Descriptions of the pins are listed in [Table 3-1](#) (single op amps) and [Table 3-2](#) (dual and quad op amps).

TABLE 3-1: PIN FUNCTION TABLE FOR SINGLE OP AMPS

MCP601		MCP601R	MCP603		Symbol	Description
PDIP, SOIC, TSSOP	SOT-23-5 (Note 1)	SOT-23-5	SOT-23-6	PDIP, SOIC, TSSOP		
6	1	1	6	6	V_{OUT}	Analog Output
2	4	4	2	2	V_{IN^-}	Inverting Input
3	3	3	3	3	V_{IN^+}	Non-inverting Input
7	5	2	7	7	V_{DD}	Positive Power Supply
4	2	5	4	4	V_{SS}	Negative Power Supply
—	—	—	8	8	\overline{CS}	Chip Select
1, 5, 8	—	—	1, 5	1	NC	No Internal Connection

Note 1: The MCP601R is only available in the 5-pin SOT-23 package.

TABLE 3-2: PIN FUNCTION TABLE FOR DUAL AND QUAD OP AMPS

MCP602	MCP604	Symbol	Description
PDIP, SOIC, TSSOP	PDIP, SOIC, TSSOP		
1	1	V_{OUTA}	Analog Output (op amp A)
2	2	V_{INA^-}	Inverting Input (op amp A)
3	3	V_{INA^+}	Non-inverting Input (op amp A)
8	4	V_{DD}	Positive Power Supply
5	5	V_{INB^+}	Non-inverting Input (op amp B)
6	6	V_{INB^-}	Inverting Input (op amp B)
7	7	V_{OUTB}	Analog Output (op amp B)
—	8	V_{OUTC}	Analog Output (op amp C)
—	9	V_{INC^-}	Inverting Input (op amp C)
—	10	V_{INC^+}	Non-inverting Input (op amp C)
4	11	V_{SS}	Negative Power Supply
—	12	V_{IND^+}	Non-inverting Input (op amp D)
—	13	V_{IND^-}	Inverting Input (op amp D)
—	14	V_{OUTD}	Analog Output (op amp D)

3.1 Analog Outputs

The op amp output pins are low-impedance voltage sources.

3.2 Analog Inputs

The op amp non-inverting and inverting inputs are high-impedance CMOS inputs with low bias currents.

3.3 Chip Select Digital Input

This is a CMOS, Schmitt-triggered input that places the part into a low power mode of operation.

3.4 Power Supply Pins

The positive power supply pin (V_{DD}) is 2.5V to 6.0V higher than the negative power supply pin (V_{SS}). For normal operation, the other pins are at voltages between V_{SS} and V_{DD} .

Typically, these parts are used in a single (positive) supply configuration. In this case, V_{SS} is connected to ground and V_{DD} is connected to the supply. V_{DD} will need bypass capacitors.

MCP601/1R/2/3/4

4.0 APPLICATIONS INFORMATION

The MCP601/1R/2/3/4 family of op amps are fabricated on Microchip's state-of-the-art CMOS process. They are unity-gain stable and suitable for a wide range of general purpose applications.

4.1 Inputs

4.1.1 PHASE REVERSAL

The MCP601/1R/2/3/4 op amp is designed to prevent phase reversal when the input pins exceed the supply voltages. [Figure 2-34](#) shows the input voltage exceeding the supply voltage without any phase reversal.

4.1.2 INPUT VOLTAGE AND CURRENT LIMITS

The ESD protection on the inputs can be depicted as shown in [Figure 4-1](#). This structure was chosen to protect the input transistors, and to minimize input bias current (I_B). The input ESD diodes clamp the inputs when they try to go more than one diode drop below V_{SS} . They also clamp any voltages that go too far above V_{DD} ; their breakdown voltage is high enough to allow normal operation, and low enough to bypass quick ESD events within the specified limits.

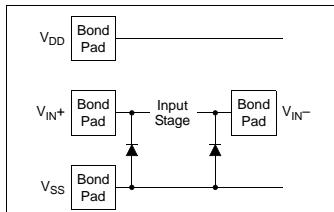


FIGURE 4-1: Simplified Analog Input ESD Structures.

In order to prevent damage and/or improper operation of these op amps, the circuit they are in must limit the currents and voltages at the V_{IN^+} and V_{IN^-} pins (see [Absolute Maximum Ratings](#) † at the beginning of [Section 1.0 "Electrical Characteristics"](#)). [Figure 4-2](#) shows the recommended approach to protecting these inputs. The internal ESD diodes prevent the input pins (V_{IN^+} and V_{IN^-}) from going too far below ground, and the resistors R_1 and R_2 limit the possible current drawn out of the input pins. Diodes D_1 and D_2 prevent the input pins (V_{IN^+} and V_{IN^-}) from going too far above V_{DD} , and dump any currents onto V_{DD} . When implemented as shown, resistors R_1 and R_2 also limit the current through D_1 and D_2 .

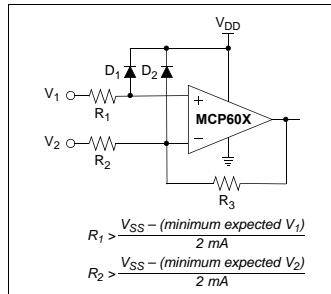


FIGURE 4-2: Protecting the Analog Inputs.

It is also possible to connect the diodes to the left of resistors R_1 and R_2 . In this case, current through the diodes D_1 and D_2 needs to be limited by some other mechanism. The resistors then serve as in-rush current limiters; the DC current into the input pins (V_{IN^+} and V_{IN^-}) should be very small.

A significant amount of current can flow out of the inputs when the common mode voltage (V_{CM}) is below ground (V_{SS}); see [Figure 2-34](#). Applications that are high impedance may need to limit the useable voltage range.

4.1.3 NORMAL OPERATION

The Common Mode Input Voltage Range (V_{CMR}) includes ground in single-supply systems (V_{SS}), but does not include V_{DD} . This means that the amplifier input behaves linearly as long as the Common Mode Input Voltage (V_{CM}) is kept within the specified V_{CMR} limits ($V_{SS}-0.3V$ to $V_{DD}-1.2V$ at $+25^\circ C$).

[Figure 4-3](#) shows a unity gain buffer. Since V_{OUT} is the same voltage as the inverting input, V_{OUT} must be kept below $V_{DD}-1.2V$ for correct operation.

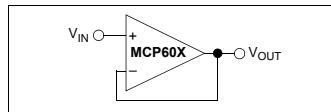


FIGURE 4-3: Unity Gain Buffer has a Limited V_{OUT} Range.

MCP601/1R/2/3/4

4.2 Rail-to-Rail Output

There are two specifications that describe the output swing capability of the MCP601/1R/2/3/4 family of op amps. The first specification (Maximum Output Voltage Swing) defines the absolute maximum swing that can be achieved under the specified load conditions. For instance, the output voltage swings to within 15 mV of the negative rail with a 25 k Ω load to $V_{DD}/2$. [Figure 2-33](#) shows how the output voltage is limited when the input goes beyond the linear region of operation.

The second specification that describes the output swing capability of these amplifiers is the Linear Output Voltage Swing. This specification defines the maximum output swing that can be achieved while the amplifier is still operating in its linear region. To verify linear operation in this range, the large signal (DC Open-Loop Gain (A_{OL})) is measured at points 100 mV inside the supply rails. The measurement must exceed the specified gains in the specification table.

4.3 MCP603 Chip Select

The MCP603 is a single amplifier with Chip Select (CS). When CS is pulled high, the supply current drops to -0.7 μ A (typ.), which is pulled through the CS pin to V_{SS} . When this happens, the amplifier output is put into a high-impedance state. Pulling CS low enables the amplifier.

The CS pin has an internal 5 M Ω (typical) pull-down resistor connected to V_{SS} , so it will go low if the CS pin is left floating. [Figure 1-1](#) is the Chip Select timing diagram and shows the output voltage, supply currents, and CS current in response to a CS pulse. [Figure 2-27](#) shows the measured output voltage response to a CS pulse.

4.4 Capacitive Loads

Driving large capacitive loads can cause stability problems for voltage feedback op amps. As the load capacitance increases, the feedback loop's phase margin decreases and the closed-loop bandwidth is reduced. This produces gain peaking in the frequency response with overshoot and ringing in the step response.

When driving large capacitive loads with these op amps (e.g., > 40 pF when $G = +1$), a small series resistor at the output (R_{ISO} in [Figure 4-4](#)) improves the feedback loop's phase margin (stability) by making the output load resistive at higher frequencies. The bandwidth will be generally lower than the bandwidth with no capacitive load.

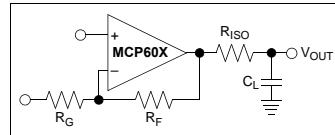


FIGURE 4-4: Output resistor R_{ISO} stabilizes large capacitive loads.

[Figure 4-5](#) gives recommended R_{ISO} values for different capacitive loads and gains. The x-axis is the normalized load capacitance (C_L/G_N) in order to make it easier to interpret the plot for arbitrary gains. G_N is the circuit's noise gain. For non-inverting gains, G_N and the gain are equal. For inverting gains, $G_N = 1 + |\text{Gain}|$ (e.g., -1 V/V gives $G_N = +2$ V/V).

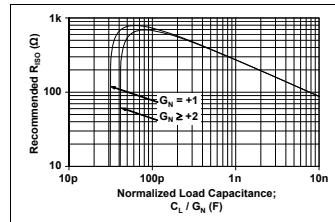


FIGURE 4-5: Recommended R_{ISO} values for capacitive loads.

Once you have selected R_{ISO} for your circuit, double-check the resulting frequency response peaking and step response overshoot in your circuit. Evaluation on the bench and simulations with the MCP601/1R/2/3/4 SPICE macro model are very helpful. Modify R_{ISO} 's value until the response is reasonable.

4.5 Supply Bypass

With this family of op amps, the power supply pin (V_{DD} for single-supply) should have a local bypass capacitor (i.e., 0.01 μ F to 0.1 μ F) within 2 mm for good high-frequency performance. It also needs a bulk capacitor (i.e., 1 μ F or larger) within 100 mm to provide large, slow currents. This bulk capacitor can be shared with nearby analog parts.

MCP601/1R/2/3/4

4.6 Unused Op Amps

An unused op amp in a quad package (MCP604) should be configured as shown in [Figure 4-6](#). These circuits prevent the output from toggling and causing crosstalk. Circuit A sets the op amp at its minimum noise gain. The resistor divider produces any desired reference voltage within the output voltage range of the op amp; the op amp buffers that reference voltage. Circuit B uses the minimum number of components and operates as a comparator, but it may draw more current.

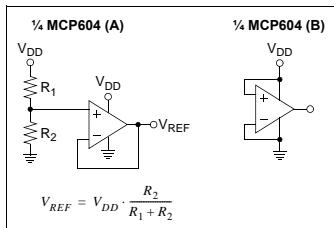


FIGURE 4-6: Unused Op Amps.

4.7 PCB Surface Leakage

In applications where low input bias current is critical, printed circuit board (PCB) surface leakage effects need to be considered. Surface leakage is caused by humidity, dust or other contamination on the board. Under low humidity conditions, a typical resistance between nearby traces is $10^{12}\Omega$. A 5 V difference would cause 5 pA of current to flow. This is greater than the MCP601/1R/2/3/4 family's bias current at $+25^\circ\text{C}$ (1 pA, typical).

The easiest way to reduce surface leakage is to use a guard ring around sensitive pins (or traces). The guard ring is biased at the same voltage as the sensitive pin. An example of this type of layout is shown in [Figure 4-7](#).

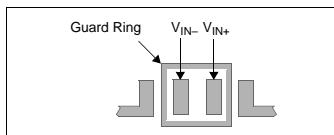


FIGURE 4-7: Example Guard Ring layout.

1. Connect the guard ring to the inverting input pin (V_{IN^-}) for non-inverting gain amplifiers, including unity-gain buffers. This biases the guard ring to the common mode input voltage.

2. Connect the guard ring to the non-inverting input pin (V_{IN^+}) for inverting gain amplifiers and transimpedance amplifiers (converts current to voltage, such as photo detectors). This biases the guard ring to the same reference voltage as the op amp (e.g., $V_{DD}/2$ or ground).

4.8 Typical Applications

4.8.1 ANALOG FILTERS

[Figure 4-8](#) and [Figure 4-9](#) show low-pass, second-order, Butterworth filters with a cutoff frequency of 10 Hz. The filter in [Figure 4-8](#) has a non-inverting gain of +1 V/V, and the filter in [Figure 4-9](#) has an inverting gain of -1 V/V.

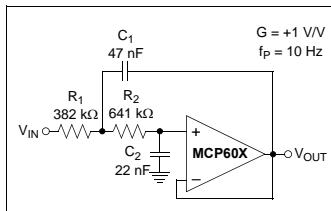


FIGURE 4-8: Second-Order, Low-Pass Sallen-Key Filter.

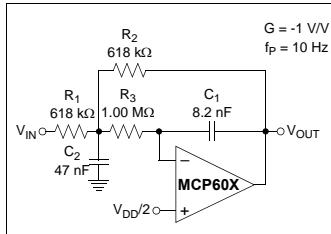


FIGURE 4-9: Second-Order, Low-Pass Multiple-Feedback Filter.

The MCP601/1R/2/3/4 family of op amps have low input bias current, which allows the designer to select larger resistor values and smaller capacitor values for these filters. This helps produce a compact PCB layout. These filters, and others, can be designed using Microchip's Design Aids; see [Section 5.2 "FilterLab® Software"](#) and [Section 5.3 "Mindi™ Simulator Tool"](#).

MCP601/1R/2/3/4

4.8.2 INSTRUMENTATION AMPLIFIER CIRCUITS

Instrumentation amplifiers have a differential input that subtracts one input voltage from another and rejects common mode signals. These amplifiers also provide a single-ended output voltage.

The three-op amp instrumentation amplifier is illustrated in Figure 4-10. One advantage of this approach is unity-gain operation, while one disadvantage is that the common mode input range is reduced as R_2/R_G gets larger.

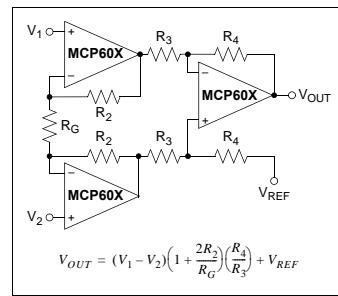


FIGURE 4-10: Three-Op Amp Instrumentation Amplifier.

The two-op amp instrumentation amplifier is shown in Figure 4-11. While its power consumption is lower than the three-op amp version, its main drawbacks are that the common mode range is reduced with higher gains and it must be configured in gains of two or higher.

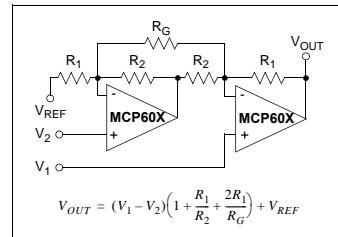


FIGURE 4-11: Two-Op Amp Instrumentation Amplifier.

Both instrumentation amplifiers should use a bulk bypass capacitor of at least 1 μ F. The CMRR of these amplifiers will be set by both the op amp CMRR and resistor matching.

4.8.3 PHOTO DETECTION

The MCP601/1R/2/3/4 op amps can be used to easily convert the signal from a sensor that produces an output current (such as a photo diode) into a voltage (a transimpedance amplifier). This is implemented with a single resistor (R_2) in the feedback loop of the amplifiers shown in Figure 4-12 and Figure 4-13. The optional capacitor (C_2) sometimes provides stability for these circuits.

A photodiode configured in the Photovoltaic mode has zero voltage potential placed across it (Figure 4-12). In this mode, the light sensitivity and linearity is maximized, making it best suited for precision applications. The key amplifier specifications for this application are: low input bias current, low noise, common mode input voltage range (including ground), and rail-to-rail output.

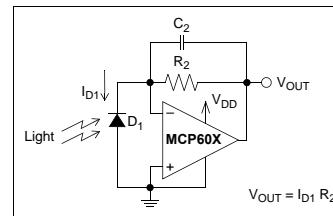


FIGURE 4-12: Photovoltaic Mode Detector.

In contrast, a photodiode that is configured in the Photoconductive mode has a reverse bias voltage across the photo-sensing element (Figure 4-13). This decreases the diode capacitance, which facilitates high-speed operation (e.g., high-speed digital communications). The design trade-off is increased diode leakage current and linearity errors. The op amp needs to have a wide Gain Bandwidth Product (GBWP).

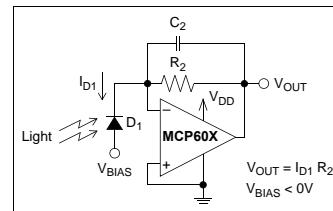


FIGURE 4-13: Photoconductive Mode Detector.

MCP601/1R/2/3/4

5.0 DESIGN AIDS

Microchip provides the basic design tools needed for the MCP601/1R/2/3/4 family of op amps.

5.1 SPICE Macro Model

The latest SPICE macro model for the MCP601/1R/2/3/4 op amps is available on the Microchip web site at www.microchip.com. This model is intended to be an initial design tool that works well in the op amp's linear region of operation over the temperature range. See the model file for information on its capabilities.

Bench testing is a very important part of any design and cannot be replaced with simulations. Also, simulation results using this macro model need to be validated by comparing them to the data sheet specifications and characteristic curves.

5.2 FilterLab® Software

Microchip's FilterLab® software is an innovative software tool that simplifies analog active filter (using op amps) design. Available at no cost from the Microchip web site at www.microchip.com/filterlab, the FilterLab design tool provides full schematic diagrams of the filter circuit with component values. It also outputs the filter circuit in SPICE format, which can be used with the macro model to simulate actual filter performance.

5.3 Mindi™ Simulator Tool

Microchip's Mindi™ simulator tool aids in the design of various circuits useful for active filter, amplifier and power-management applications. It is a free online simulation tool available from the Microchip web site at www.microchip.com/mindi. This interactive simulator enables designers to quickly generate circuit diagrams, simulate circuits. Circuits developed using the Mindi simulation tool can be downloaded to a personal computer or workstation.

5.4 MAPS (Microchip Advanced Part Selector)

MAPS is a software tool that helps semiconductor professionals efficiently identify Microchip devices that fit a particular design requirement. Available at no cost from the Microchip website at www.microchip.com/maps, the MAPS is an overall selection tool for Microchip's product portfolio that includes Analog, Memory, MCUs and DSCs. Using this tool you can define a filter to sort features for a parametric search of devices and export side-by-side technical comparison reports. Helpful links are also provided for Datasheets, Purchase, and Sampling of Microchip parts.

5.5 Analog Demonstration and Evaluation Boards

Microchip offers a broad spectrum of Analog Demonstration and Evaluation Boards that are designed to help you achieve faster time to market. For a complete listing of these boards and their corresponding user's guides and technical information, visit the Microchip web site at www.microchip.com/analogtools.

Two of our boards that are especially useful are:

- **P/N SOIC8EV: 8-Pin SOIC/MSOP/TSSOP/DIP Evaluation Board**
- **P/N SOIC14EV: 14-Pin SOIC/TSSOP/DIP Evaluation Board**

5.6 Application Notes

The following Microchip Application Notes are available on the Microchip web site at www.microchip.com/appnotes and are recommended as supplemental reference resources.

ADN003: "Select the Right Operational Amplifier for your Filtering Circuits", DS21821

AN722: "Operational Amplifier Topologies and DC Specifications", DS00722

AN723: "Operational Amplifier AC Specifications and Applications", DS00723

AN884: "Driving Capacitive Loads With Op Amps", DS00884

AN990: "Analog Sensor Conditioning Circuits – An Overview", DS00990

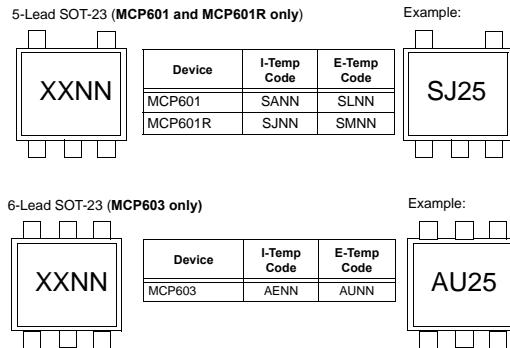
These application notes and others are listed in the design guide:

"Signal Chain Design Guide", DS21825

MCP601/1R/2/3/4

6.0 PACKAGING INFORMATION

6.1 Package Marking Information

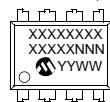


Legend:	XX...X Customer-specific information
Y	Year code (last digit of calendar year)
YY	Year code (last 2 digits of calendar year)
WW	Week code (week of January 1 is week '01')
NNN	Alphanumeric traceability code
(e3)	Pb-free JEDEC designator for Matte Tin (Sn)
*	This package is Pb-free. The Pb-free JEDEC designator (e3) can be found on the outer packaging for this package.
Note:	In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line, thus limiting the number of available characters for customer-specific information.

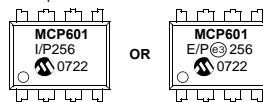
MCP601/1R/2/3/4

Package Marking Information (Continued)

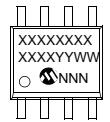
8-Lead PDIP (300 mil)



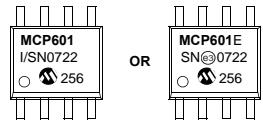
Example:



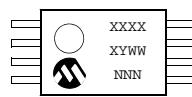
8-Lead SOIC (150 mil)



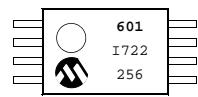
Example:

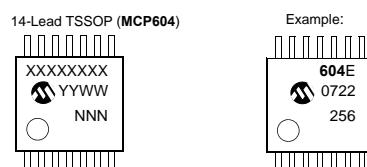
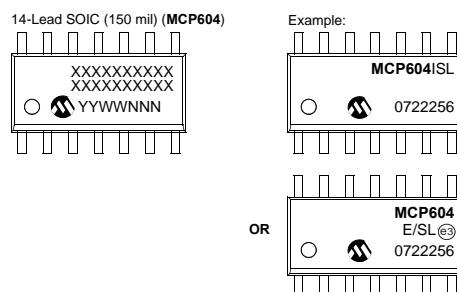
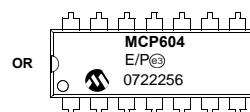
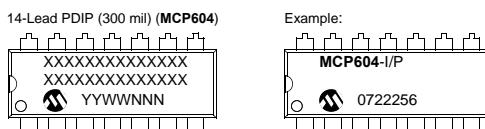


8-Lead TSSOP



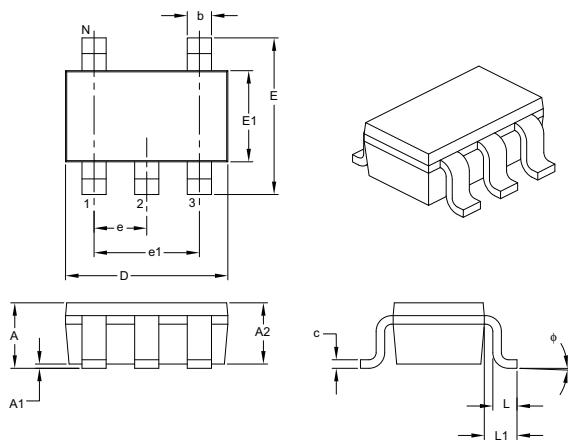
Example:



MCP601/1R/2/3/4**Package Marking Information (Continued)**

MCP601/1R/2/3/4**5-Lead Plastic Small Outline Transistor (OT) [SOT-23]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension	Limits	MILLIMETERS		
		MIN	NOM	MAX
Number of Pins	N		5	
Lead Pitch	e		0.95 BSC	
Outside Lead Pitch	e1		1.90 BSC	
Overall Height	A	0.90	—	1.45
Molded Package Thickness	A2	0.89	—	1.30
Standoff	A1	0.00	—	0.15
Overall Width	E	2.20	—	3.20
Molded Package Width	E1	1.30	—	1.80
Overall Length	D	2.70	—	3.10
Foot Length	L	0.10	—	0.60
Footprint	L1	0.35	—	0.80
Foot Angle	phi	0°	—	30°
Lead Thickness	c	0.08	—	0.26
Lead Width	b	0.20	—	0.51

Notes:

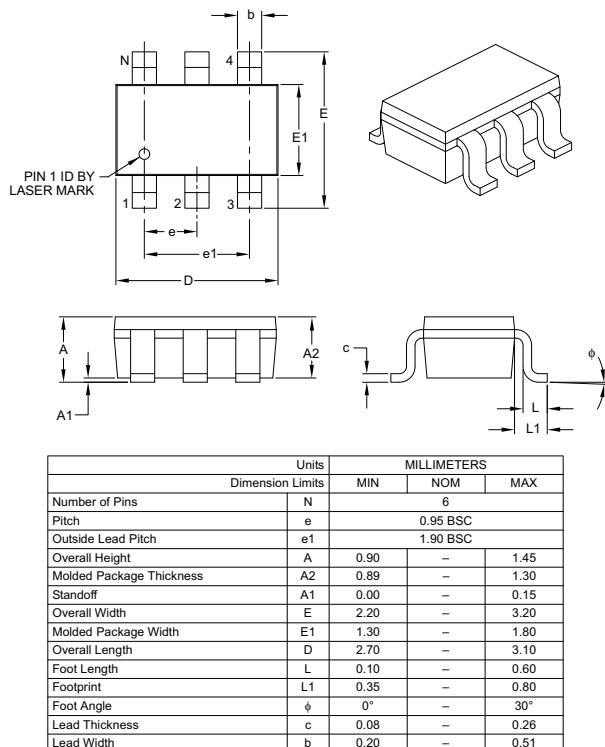
- Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.127 mm per side.
- Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-091B

MCP601/1R/2/3/4**6-Lead Plastic Small Outline Transistor (CH) [SOT-23]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>

**Notes:**

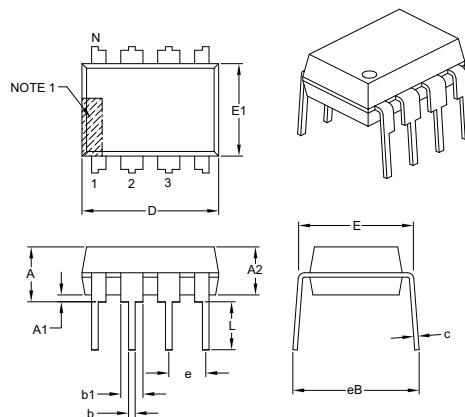
- Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.127 mm per side.
- Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-028B

MCP601/1R/2/3/4**8-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension Limits	Units			INCHES		
	MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	N	8				
Pitch	e		.100 BSC			
Top to Seating Plane	A	—	—	.210		
Molded Package Thickness	A2	.115	.130	.195		
Base to Seating Plane	A1	.015	—	—		
Shoulder to Shoulder Width	E	.290	.310	.325		
Molded Package Width	E1	.240	.250	.280		
Overall Length	D	.348	.365	.400		
Tip to Seating Plane	L	.115	.130	.150		
Lead Thickness	c	.008	.010	.015		
Upper Lead Width	b1	.040	.060	.070		
Lower Lead Width	b	.014	.018	.022		
Overall Row Spacing §	eB	—	—	.430		

Notes:

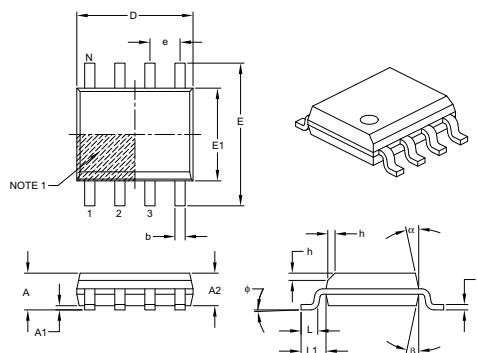
1. Pin 1 visual index feature may vary, but must be located with the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-018B

MCP601/1R/2/3/4**8-Lead Plastic Small Outline (SN) – Narrow, 3.90 mm Body [SOIC]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



		Units	MILLIMETERS		
		Dimension Limits	MIN	NOM	MAX
Number of Pins	N			8	
Pitch	e			1.27 BSC	
Overall Height	A		—	—	1.75
Molded Package Thickness	A2		1.25	—	—
Standoff §	A1		0.10	—	0.25
Overall Width	E		6.00 BSC	—	—
Molded Package Width	E1		3.90 BSC	—	—
Overall Length	D		4.90 BSC	—	—
Chamfer (optional)	h		0.25	—	0.50
Foot Length	L		0.40	—	1.27
Footprint	L1		1.04 REF	—	—
Foot Angle	φ		0°	—	8°
Lead Thickness	c		0.17	—	0.25
Lead Width	b		0.31	—	0.51
Mold Draft Angle Top	α		5°	—	15°
Mold Draft Angle Bottom	β		5°	—	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

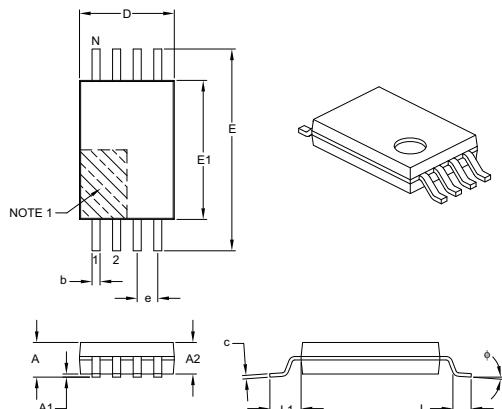
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-057B

MCP601/1R/2/3/4**8-Lead Plastic Thin Shrink Small Outline (ST) – 4.4 mm Body [TSSOP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension Limits	Units MILLIMETERS		
	MIN	NOM	MAX
Number of Pins	N	8	
Pitch	e	0.65 BSC	
Overall Height	A	—	1.20
Molded Package Thickness	A2	0.80	1.00
Standoff	A1	0.05	—
Overall Width	E	6.40 BSC	
Molded Package Width	E1	4.30	4.40
Molded Package Length	D	2.90	3.00
Foot Length	L	0.45	0.60
Footprint	L1	1.00 REF	
Foot Angle	φ	0°	8°
Lead Thickness	c	0.09	—
Lead Width	b	0.19	—
		0.30	

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

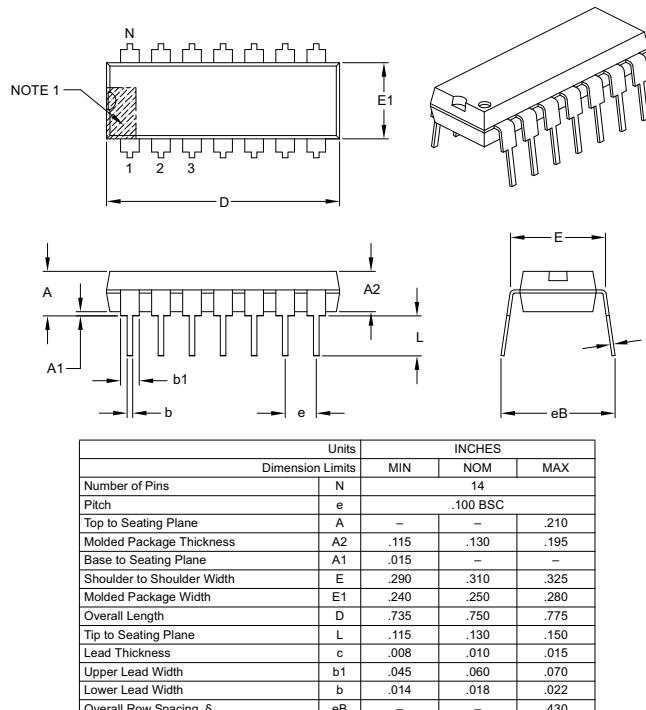
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-086B

MCP601/1R/2/3/4**14-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>

**Notes:**

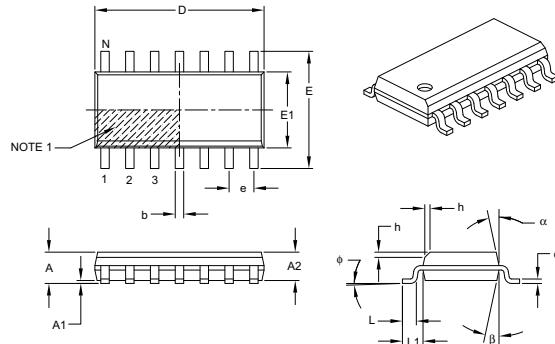
1. Pin 1 visual index feature may vary, but must be located with the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-005B

MCP601/1R/2/3/4**14-Lead Plastic Small Outline (SL) – Narrow, 3.90 mm Body [SOIC]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension	Units	MILLIMETERS		
		MIN	NOM	MAX
Number of Pins	N	14		
Pitch	e	1.27 BSC		
Overall Height	A	—	—	1.75
Molded Package Thickness	A2	1.25	—	—
Standoff §	A1	0.10	—	0.25
Overall Width	E	6.00 BSC		
Molded Package Width	E1	3.90 BSC		
Overall Length	D	8.65 BSC		
Chamfer (optional)	h	0.25	—	0.50
Foot Length	L	0.40	—	1.27
Footprint	L1	1.04 REF		
Foot Angle	φ	0°	—	8°
Lead Thickness	c	0.17	—	0.25
Lead Width	b	0.31	—	0.51
Mold Draft Angle Top	α	5°	—	15°
Mold Draft Angle Bottom	β	5°	—	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

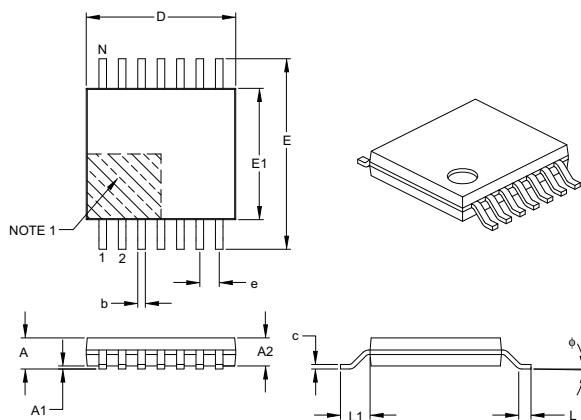
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-065B

MCP601/1R/2/3/4**14-Lead Plastic Thin Shrink Small Outline (ST) – 4.4 mm Body [TSSOP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		14	
Pitch	e		0.65 BSC	
Overall Height	A	—	—	1.20
Molded Package Thickness	A2	0.80	1.00	1.05
Standoff	A1	0.05	—	0.15
Overall Width	E		6.40 BSC	
Molded Package Width	E1	4.30	4.40	4.50
Molded Package Length	D	4.90	5.00	5.10
Foot Length	L	0.45	0.60	0.75
Footprint	L1		1.00 REF	
Foot Angle	phi	0°	—	8°
Lead Thickness	c	0.09	—	0.20
Lead Width	b	0.19	—	0.30

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-087B

MCP601/1R/2/3/4

NOTES:

MCP601/1R/2/3/4

APPENDIX A: REVISION HISTORY

Revision G (December 2007)

- Updated Figure 2-15 and Figure 2-19.
- Updated Table 3-1 and Table 3-2.
- Updated notes to **Section 1.0 "Electrical Characteristics"**.
- Expanded Analog Input Absolute Maximum Voltage Range (applies retroactively).
- Expanded operating V_{DD} to a maximum of 6.0V.
- Added Figure 2-34.
- Added **Section 4.1.1 "Phase Reversal"**, **Section 4.1.2 "Input Voltage and Current Limits"**, and **Section 4.1.3 "Normal Operation"**.
- Corrected **Section 6.0 "Packaging Information"**.

Revision F (February 2004)

- Undocumented changes.

Revision E (September 2003)

- Undocumented changes.

Revision D (April 2000)

- Undocumented changes.

Revision C (July 1999)

- Undocumented changes.

Revision B (June 1999)

- Undocumented changes.

Revision A (March 1999)

- Original Release of this Document.

MCP601/1R/2/3/4

NOTES:

MCP601/1R/2/3/4**PRODUCT IDENTIFICATION SYSTEM**

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	$\underline{-X}$	\underline{XX}	
Device	Temperature Range	Package	
Device			<p>MCP601 Single Op Amp MCP601T Single Op Amp (Tape and Reel for SOT-23, SOIC and TSSOP) MCP601RT Single Op Amp (Tape and Reel for SOT-23-5) MCP602 Dual Op Amp MCP602T Dual Op Amp (Tape and Reel for SOIC and TSSOP) MCP603 Single Op Amp with Chip Select MCP603T Single Op Amp with Chip Select (Tape and Reel for SOT-23, SOIC and TSSOP) MCP604 Quad Op Amp MCP604T Quad Op Amp (Tape and Reel for SOIC and TSSOP)</p>
Temperature Range	I		= -40°C to +85°C
	E		= -40°C to +125°C
Package	OT		= Plastic SOT-23, 5-lead (MCP601 only)
	CH		= Plastic SOT-23, 6-lead (MCP603 only)
	P		= Plastic DIP (300 mil body), 8, 14 lead
	SN		= Plastic SOIC (3.90 mm body), 8 lead
	SL		= Plastic SOIC (3.90 mm body), 14 lead
	ST		= Plastic TSSOP (4.4 mm body), 8, 14 lead

Examples:

- a) MCP601-I/P: Single Op Amp, Industrial Temperature, 8 lead PDIP package.
- b) MCP601-E/SN: Single Op Amp, Extended Temperature, 8 lead SOIC package.
- c) MCP601T-E/ST: Tape and Reel, Extended Temperature, Single Op Amp, 8 lead TSSOP package
- d) MCP601RT-I/OT: Tape and Reel, Industrial Temperature, Single Op Amp, Rotated 5 lead SOT-23 package.
- e) MCP601RT-E/OT: Tape and Reel, Extended Temperature, Single Op Amp, Rotated, 5 lead SOT-23 package.

- a) MCP602-I/SN: Dual Op Amp, Industrial Temperature, 8 lead SOIC package.
- b) MCP602-E/P: Dual Op Amp, Extended Temperature, 8 lead PDIP package.
- c) MCP602T-E/ST: Tape and Reel, Extended Temperature, Dual Op Amp, 8 lead TSSOP package.

- a) MCP603-I/SN: Industrial Temperature, Single Op Amp with Chip Select, 8 lead SOIC package.
- b) MCP603-E/P: Extended Temperature, Single Op Amp with Chip Select, 8 lead PDIP package.
- c) MCP603T-E/ST: Tape and Reel, Extended Temperature, Single Op Amp with Chip Select, 8 lead TSSOP package.
- d) MCP603T-I/SN: Tape and Reel, Industrial Temperature, Single Op Amp with Chip Select, 8 lead SOIC package.

- a) MCP604-I/P: Industrial Temperature, Quad Op Amp, 14 lead PDIP package.
- b) MCP604-E/SL: Extended Temperature, Quad Op Amp, 14 lead SOIC package.
- c) MCP604T-E/ST: Tape and Reel, Extended Temperature, Quad Op Amp, 14 lead TSSOP package.

MCP601/1R/2/3/4

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KeeLoQ, KeeLoQ logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Amplab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WipeLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SOTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2007, Microchip Technology Incorporated. Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
= ISO/TS 16949:2002 =**

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. Microchip's quality system processes and procedures are for its PIC® MCUs and PIC® MCSs, KeeLoQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2555 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471-6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2500-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

10/05/07

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both L^AT_EX and LyX:

<https://bitbucket.org/amiede/classicthesis/>