ESS Technology, Inc.

ES692
Wavetable with ROM
Data Sheet

DESCRIPTION

The ES692 is a single, highly integrated, high-performance, and economical wavetable music synthesizer for personal computers, delivering superior acoustic sound comparable to professional synthesizers. The ES692 includes reverb special effects without need of external RAM. With its embedded microcontroller, the ES692 supports General MIDI, providing for 128 melodic instruments with the ability to play back 32 voices of 16-bit data at a sampling rate of 44.1 kHz. Music is produced in high fidelity with the realism of a live symphony orchestra.

The ES692 includes a 1 MB wavetable ROM to provide a complete wavetable solution. This internal ROM provides digitally recorded sound samples of musical instruments.

The ES692 is designed to interface with the ES1xxx *Audio***Drive**® chips without requiring any glue logic or external DAC. The ES692 interfaces with the music DAC of the ES1xxx via the third serial port of the host chip, providing a cost-effective implementation of a complete wavetable music synthesizer.

Advanced power management features include suspend/ resume and automatic power-down when MIDI input is idle.

The ES692 is available in an industry-standard 100-pin Thin Quad Flat Pack (TQFP) package.

Figure 1 shows the ES692 Wavetable Music Synthesizer in conjunction with an ES1xxx *Audio***Drive**® solution to create a complete PC audio solution.

FEATURES

- Single chip, high-performance wavetable music synthesizer with embedded 1 MB ROM sample
- Reverb special effect without external RAM
- Playback of 16-bit data at 44.1 kHz via the ES1xxx DAC
- Stereo pan for each voice
- 32-voice polyphony
- MIDI serial port compatible with MPU-401 serial port of the ES1xxx
- General MIDI instrument set – 128 melodic and 47 rhythm instruments
- Digital serial interface with the ES1xxx
- Glueless interface with an ES1xxx *Audio***Drive**® chip
- Advanced power management with automatic power-down when MIDI input is idle
- Context upload/download for suspend/resume
- 100-pin TQFP package

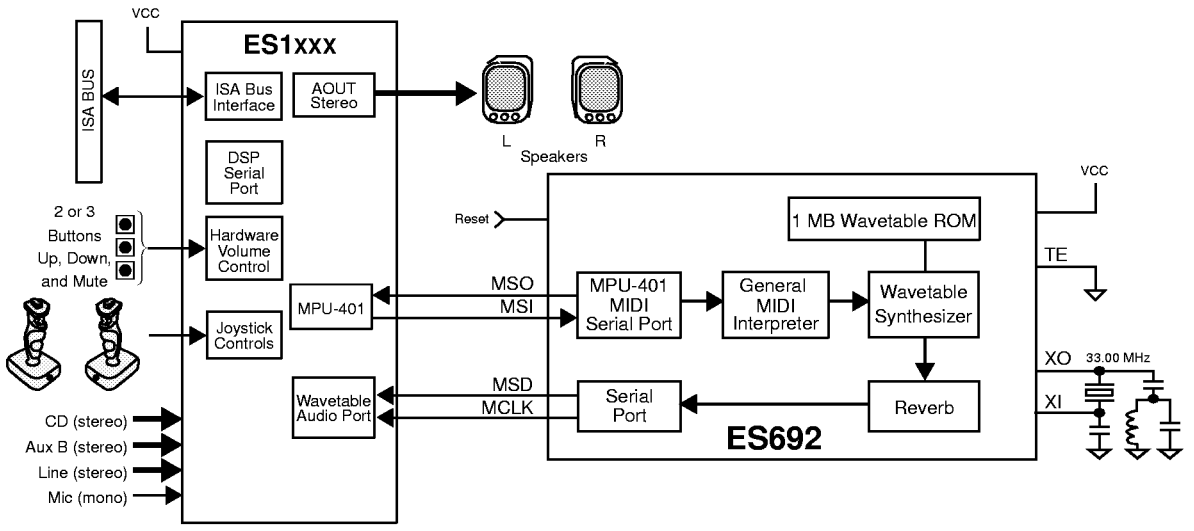


Figure 1 ES692 Wavetable Solution with ES1xxx

CONTENTS

FIGURES

TABLES

PINOUT

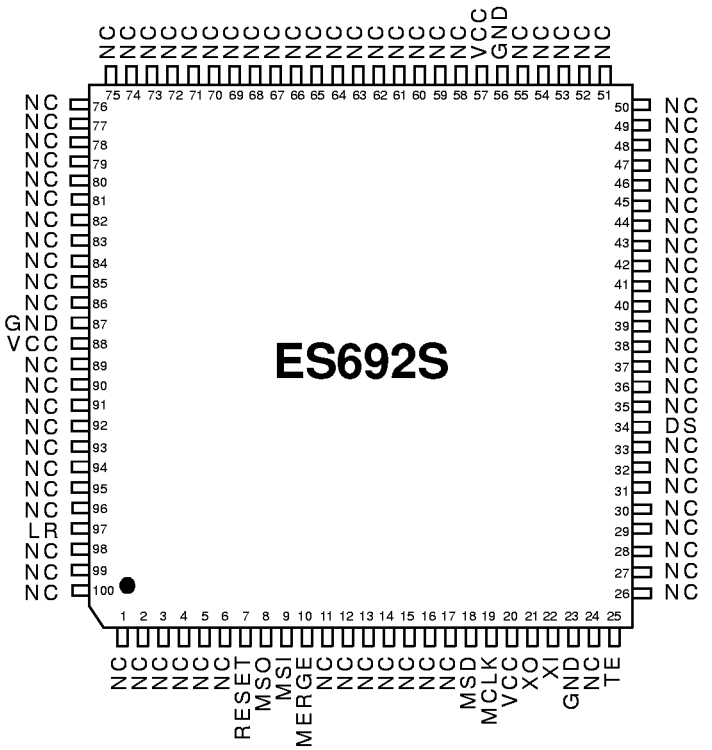


Figure 2 ES692 Pinout

PIN DESCRIPTION

Name	Number	I/O	Definition
RESET	7	I	Active-high reset input.
MSO	8	O	MIDI serial output for two-way connection to ES1xxx *Audio***Drive®** controller.
MSI	9	I	MIDI serial input from ES1xxx *Audio***Drive®** controller.
MERGE	10	I	For one-way MIDI connection, this pin is left no-connect. For two-way MIDI connection, this pin is connected to external MIDI input. Use external pull-up resistor when MERGE is not connected to MIDI input. Normally, this pin is internally connected to MSO pin.
MSD	18	O	Music serial data to the ES1xxx *Audio***Drive®** controller.
MCLK	19	O	Music serial clock to the ES1xxx *Audio***Drive®** controller.
VCC	20, 57, 88	I	Power supply voltage (3.3 V).
XO	21	O	Oscillator output. Connect to 33.0 MHz crystal.
XI	22	I	Oscillator input. Connect to 33.0 MHz crystal.
GND	23, 56, 87	I	Ground.
TE	25	I	Test pin (reserved). Connect to GND for proper operation.
DS	34	I	Data format select for audio serial port. 0: 2-wire interface to ES1xxx *Audio***Drive®** controller 1: 3-wire interface to stereo DAC
LR	97	O	Left/right strobe for 3-wire interface to stereo DAC.
NC	6:1, 17:11, 24, 33:26, 55:35, 86:58, 96:89, 100:98		No connection.

FUNCTIONAL DESCRIPTION

Figure 3 shows the internal architecture of the ES692 Wavetable Synthesizer, including audio and control signal inputs and outputs. MIDI commands are received by the chip's buffered MIDI Serial Port (MSI pin).

The General MIDI Interpreter functional block inspects each MIDI command passed to it by the MIDI Serial Port and transfers it to the next appropriate functional block. The Interpreter informs the Synthesizer Control Unit which sound sample to access.

The Synthesizer Control Unit locates the requested sound sample in the internal wavetable ROM, informing the Synthesis Unit block that the sample will be transmitted through the MPU-401 Serial Port. The Synthesizer Control Unit can be programmed to transmit an ID string to the ES1xxx upon request from the *Audio***Drive**® chip.

Sound sample data is shifted out on MIDI Serial Data pin 18, under control of the Serial Port's Bit Clock output pin 19, for D/A conversion by ES1xxx MPU-401 MIDI serial input.

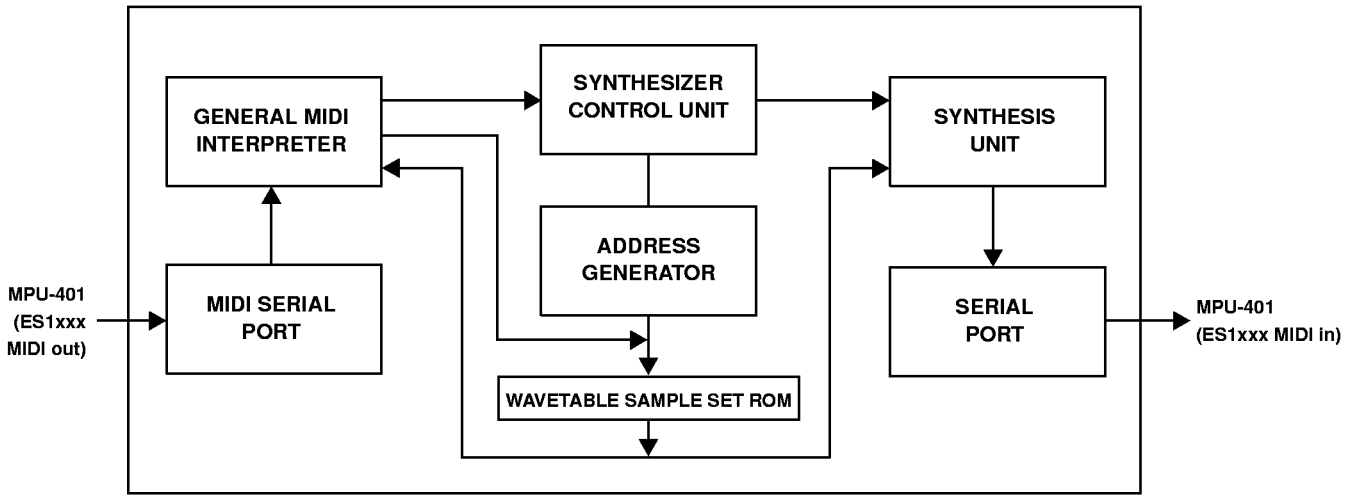


Figure 3 ES692 Block Diagram

MIDI IMPLEMENTATION

A proper MIDI implementation of the ES692 requires connections for receiving the MIDI MPU-401 standard set-up described below:

- MIDI Channels. 1 to 16 channels; Channel 10 is the percussion channel.
- Mode. Third mode: Omni Off and Poly.
- Number of Notes. 0 to 127 recognized, 12 to 108 True Voice.
- Velocity. Note ON.
- After-touch. Not available.
- Pitch Bend. Default range is ±2 semitones.
- Controller 1. Modulation controller.
- Controller 6. Data entry MSB.
- Controller 7. Channel volume.
- Controller 10. Pan (except drums).
- Controller 11. Expression.
- Controller 64. Hold pedal.
- Controllers 100, 101. RPN LSB, MSB.
- Controller 120. All sounds off.
- Controller 121. Reset all controllers.
- Controller 123. All notes off.
- Program Change. 0 to 127, or programs 1 to 128.
- System Exclusive. General MIDI on. Same as resetting all controllers.
- System Common. Not available.
- System Real Time. Not available.

A typical MIDI implementation is shown in Figure 1 on the first page. A connection scheme between the ES692 and ES1xxx *Audio***Drive**® controller is described later in this data sheet.

MIDI Implementation Chart

Table 1 ES692 MIDI Implementation Chart

Function		Transmitted	Received	Remarks
MIDI channels		No	1-16	Channel 10 is the percussion channel.
Mode		No	3	Omni off, poly
Note number	Recognized	No	0-127	
	True Voice	No	12-108	
Velocity	Note ON	No	Yes	
	Note OFF	No	No	
Aftertouch	Key	No	No	
	Channel	No	No	
Pitch bend		No	Yes	Default range: ±2 semitones
Controllers	1	No	Yes	Modulation controller
	6	No	Yes	Data entry MSB
	7	No	Yes	Channel volume
	10	No	Yes	Pan (except drums)
	11	No	Yes	Expression
	64	No	Yes	Hold pedal
	91	No	Yes	Reverb
	100,101	No	Yes	RPN LSB, MSB
	120	No	Yes	All sounds off
	121	No	Yes	Reset all controllers
	123	No	Yes	All notes off
Program change		No	0-127	Programs 1-128
System exclusive		No	General MIDI ON	Same as reset all controllers
System common		No	No	
System real time		No	No	

General MIDI Sound Set

The ES692's MIDI sound set is shown in Table 2, listed by instrument group and group program numbers.

Table 3 shows the complete sound set, by instrument and instrument program number, available using all channels except Channel 10, which is mapped out in Figure 4. Note that the transmitted program numbers 0 to 127 correspond to the MIDI standard 1 to 128.

Table 2 General MIDI Sound Set Groupings

Program Numbers	As Transmitted	Instrument Group
1 - 8	0 - 7	Piano
9 - 16	8 - 15	Chromatic Percussion
17 - 24	16 - 23	Organ
25 - 32	24 - 31	Guitar
33 - 40	32 - 39	Bass
41 - 48	40 - 47	Strings
49 - 56	46 - 55	Ensemble
57 - 64	56 - 63	Brass
65 - 72	64 - 71	Reed
73 - 80	72 - 79	Pipe
81 - 88	80 - 87	Synth Lead
89 - 96	88 - 95	Synth Pad
97 - 104	96 - 103	Synth Effects
105 - 112	104 - 111	Ethnic
113 - 120	112 - 119	Percussive
121 - 128	120 - 127	Sound Effects

Table 3 General MIDI Sound Set

Prog	Instrument	Prog	Instrument	Prog	Instrument	Prog	Instrument
1	Acoustic Grand Piano	33	Acoustic Bass	65	Soprano Sax	97	FX 1 (Rain)
2	Bright Acoustic Piano	34	Electric Bass (Finger)	66	Alto Sax	98	FX 2 (Soundtrack)
3	Electric Grand Piano	35	Electric Bass (Pick)	67	Tenor Sax	99	FX 3 (Crystal)
4	Honky-tonk Piano	36	Fretless Bass	68	Baritone Sax	100	FX 4 (Atmosphere)
5	Electric Piano 1	37	Slap Bass 1	69	Oboe	101	FX 5 (Brightness)
6	Electric Piano 2	38	Slap Bass 2	70	English Horn	102	FX 6 (Goblins)
7	Harpsichord	39	Synth Bass 1	71	Bassoon	103	FX 7 (Echoes)
8	Clavinet	40	Synth Bass 2	72	Clarinet	104	FX 8 (Sci-Fi)
9	Celesta	41	Violin	73	Piccolo	105	Sitar
10	Glockenspiel	42	Viola	74	Flute	106	Banjo
11	Music Box	43	Cello	75	Recorder	107	Shamisen
12	Vibraphone	44	Contrabass	76	Pan Flute	108	Koto
13	Marimba	45	Tremolo Strings	77	Blown Bottle	109	Kalimba
14	Xylophone	46	Pizzicato Strings	78	Shakuhachi	110	Bagpipe
15	Tubular Bells	47	Orchestral Harp	79	Whistle	111	Fiddle
16	Dulcimer	48	Timpani	80	Ocarina	112	Shanai
17	Drawbar Organ	49	String Ensemble 1	81	Lead 1 (Square)	113	Tinkle Bell
18	Percussive Organ	50	String Ensemble 2	82	Lead 2 (Sawtooth)	114	Agogo
19	Rock Organ	51	Synth Strings 1	83	Lead 3 (Calliope)	115	Steel Drums
20	Church Organ	52	Synth Strings 2	84	Lead 4 (Chiff)	116	Woodblock
21	Reed Organ	53	Choir Aahs	85	Lead 5 (Charang)	117	Taiko Drum
22	Accordion	54	Voice Oohs	86	Lead 6 (Voice)	118	Melodic Tom
23	Harmonica	55	Synth Voice	87	Lead 7 (Fifths)	119	Synth Drum
24	Tango Accordion	56	Orchestra Hit	88	Lead 8 (Bass + Lead)	120	Reverse Cymbal
25	Acoustic Guitar (Nylon)	57	Trumpet	89	Pad 1 (New Age)	121	Guitar Fret Noise
26	Acoustic Guitar(Steel)	58	Trombone	90	Pad 2 (Warm)	122	Breath Noise
27	Electric Guitar (Jazz)	59	Tuba	91	Pad 3 (Polysynth)	123	Seashore
28	Electric Guitar (Clean)	60	Muted Trumpet	92	Pad 4 (Choir)	124	Bird Tweet
29	Electric Guitar (Muted)	61	French Horn	93	Pad 5 (Bowed)	125	Telephone Ring
30	Overdriven Guitar	62	Brass Section	94	Pad 6 (Metallic)	126	Helicopter
31	Distortion Guitar	63	Synth Brass 1	95	Pad 7 (Halo)	127	Applause
32	Guitar Harmonics	64	Synth Brass 2	96	Pad 8 (Sweep)	128	Gunshot

35	Acoustic Bass Drum
36	Bass Drum
37	Side Stick
38	Acoustic Snare
39	Hand Clap
40	Electric Snare
41	Low Floor Tom
42	Closed Hi-Hat
43	High Floor Tom
44	Pedal Hi-Hat
45	Low Tom
46	Open Hi-Hat
47	Low Mid Tom
48	Hi Mid Tom
49	Crash Cymbal 1
50	High Tom
51	Ride Cymbal 1
52	Chinese Cymbal
53	Ride Bell
54	Tambourine
55	Splash Cymbal
56	Cowbell
57	Crash Cymbal 2
58	Vibraslap
59	Ride Cymbal 2
60	Hi Bongo
61	Low Bongo
62	Mute Hi Conga
63	Open Hi Conga
64	Low Conga
65	High Timbale
66	Low Timbale
67	High Agogo
68	Low Agogo
69	Cabasa
70	Maracas
71	Short Whistle
72	Long Whistle
73	Short Guiro
74	Long Guiro
75	Claves
76	Hi Woodblock
77	Low Woodblock
78	Mute Cuica
79	Open Cuica
80	Mute Triangle
81	Open Triangle

Figure 4 General MIDI Percussion Map (Channel 10)

One-Way MIDI Interface

Figure 5 shows the simplest MIDI interface between the ES692 and ES1xxx chips. MIDI data is transmitted from the MPU-401 port to the ES692, but cannot be received from the ES692. This prevents the host processor from detecting the presence of the ES692. It also prevents using the context upload capability for suspend/resume. This is the easiest connection method if these two drawbacks are not serious, and it works especially well when the ES692 is an optional daughter card.

The ES1xxx MPU-401 transmit and receive wires share a DB15 connector with a joystick (not shown). A special adapter cable that splits the joystick from the MIDI must be used. The 2.2k ohm resistors protect against static electricity.

Note that, from the point of view of the MPU-401 transmitter, the ES692 is in parallel with any external synthesizers.

The ES1xxx's MPU-401 serial port can be in one of two modes: "Smart" or "UART." When transmitting to the ES692 or receiving MIDI from an external source, the port is always in UART mode. When not in use, the port may be left in Smart mode, in which all MIDI data coming into the MPU-401 port's MSI input is automatically echoed back out the MSO output. Thus, in Smart mode, an external keyboard would be able to transmit indirectly to the ES692 through the MPU-401 port.

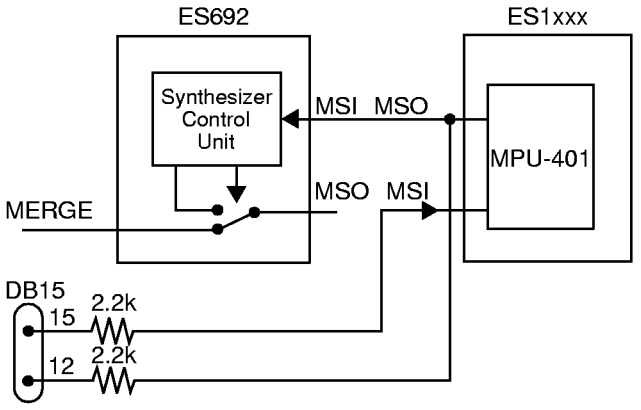


Figure 5 One-Way MIDI Interface

Two-Way MIDI Interface

With a two-way interface like the one shown in Figure 6, the ES692 can transmit data back to the ES1xxx. The host processor then commands the ES692 to send back an ID string (see Table 4) to detect the presence of the ES692. It can also ask for a context upload for suspend/resume.

Normally, the ES692's MERGE input is passed through to its MSO output. When the Synthesizer Control Unit must transmit back to the ES1xxx, it controls a switch that enables transmit.

Before switching the MSO output away from MERGE, the controller waits for inactivity on the MERGE input (i.e., more than 10 serial bits are high). The host system software is responsible for discarding any MIDI data received from the MERGE pin that comes in before the controller is able to access the MSO output.

After completing its transmission, the controller restores the MERGE-to-MSO connection and waits for a period of inactivity on the MERGE pin before switching.

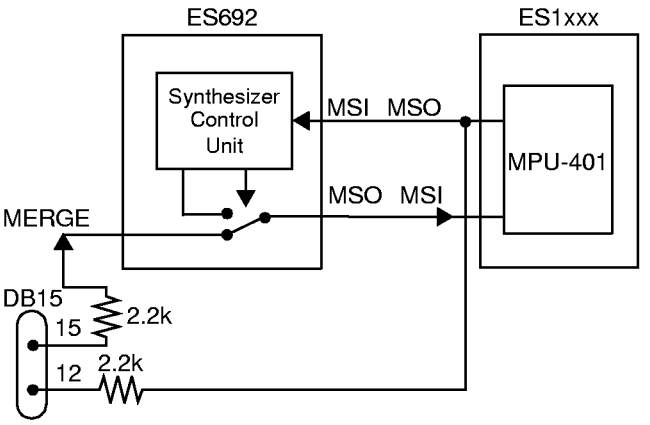


Figure 6 Two-Way MIDI Interface

System Exclusive Messages

MIDI provides a command, called System Exclusive, that allows messages that are unique to a single manufacturer. Other MIDI devices are guaranteed to ignore System Exclusive messages with different manufacturer ID numbers. The ES692 manufacturer ID is the following three-byte sequence: 00h, 00h, 7Bh.

The format of a System Exclusive message is:

F0h <id bytes><command>[<data>]F7h

A list of System Exclusive commands currently supported is shown in Table 4 below.

Table 4 Sysex Commands

Sysex Command	Function
0	Return ID string. Causes the ES692 to transmit a message as: F0 00 00 7B “ES692nn”... Where “nn” is the version number in ASCII. Up to 32 bytes follow the version number. These bytes must be read and discarded. The ID string is not guaranteed to contain F7h at the end of the ID string. The recommended procedure is to have a loop to read up to 32 bytes after the version number. The read loop should have a short time-out of about 1 millisecond between bytes received.
1	Reset ES692. Restores all initial conditions.
3	Disable activity detection. Activity detection is the mode in which the ES692 stops the MCLK output after 5 seconds of not receiving any data on its MSI pin.
4	Enable activity detection. Activity detection is the mode in which the ES692 stops the MCLK output after 5 seconds of not receiving any data on its MSI pin. This enable command has no effect if the ES692 is operating in Serial mode.
5	Disable automatic power-down when MSI input is idle. This is the reset default.
6	Enable automatic power-down when MSI input is idle for 30 seconds. The MSI pin must be continuously high for 30 seconds to cause the ES692 to enter power-down state. The oscillator continues to run. The operating current is typically reduced to 0.5 mA. The ES692 wakes when MSI goes low.
7	Power-down command. This command puts the ES692 to upload in a low-power state. The oscillator continues to run. The ES692 wakes from the low-power state if the MSI goes low.
8	Suspend request. This command causes the ES692 to upload its current context.
9	Resume command. This command causes the ES692 to download its current context after power is restored.

DESIGN CONSIDERATIONS

Audio Serial Port Mode 0

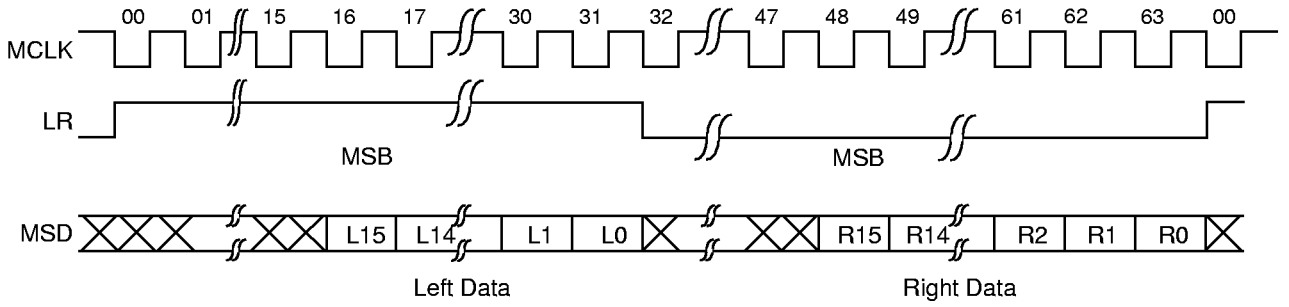
The ES692 does not require an external DAC to interface to the ES1xxx *Audio***Drive®** chip. ESS *Audio***Drive®** chips allow the ES692 to automatically acquire the DAC that is normally used by the internal FM synthesizer. In this mode, connect input pin DS to GND.

In order for the ES692 to acquire the FM DAC, bit 4 of mixer register 48h inside the ES1xxx must be set high. When bit 4 is set high, activity on the ES692 MCLK signal causes the FM DAC to be connected to the ES692. If MCLK stays low for more than a few sample periods the ES1xxx reconnects the FM DAC to the FM synthesizer.

After reset, the ES692 transmits samples continuously. In this mode, bit 4 of mixer register 48h must be set/cleared to assign the current owner of the FM DAC. The ES692 can be programmed to enter Activity Detect mode, using System Exclusive command 4, in which the ES692 blocks the serial port output (i.e., sets MCLK low) if no MIDI input is detected on the MSI pin for a period of 5 seconds. The ES692 then resumes output of serial port data as soon as MIDI input is detected on the MSI pin.

Audio Serial Port Mode 1

Audio Serial Port Mode 1 is intended for use with a third-party stereo DAC. Connect input pin DS high to VCC. Figure 7 below is a timing diagram for Audio Serial Port Mode 1.



Bit Clock Rate (MCLK):	2.75 MHz
Sample Rate (LR):	42968.75 Hz
MCLK Clocks per Sample:	64
MSD Format:	16 bits, 2's complement, MSB first

MSD and LR change after falling edge of MCLK. Hold time relative to MCLK falling edge is 0-25 nsec.

Figure 7 Audio Serial Port Mode 1 Timing

Choosing the Oscillator Crystal

The ES692 requires a 33.000 MHz crystal. Either a load capacitance or series-type crystal can be used. The load capacitance is determined by the series combination capacitance on oscillator pins XI and XO. If both XI and XO have 10 pF capacitors, plus 5 pF of stray capacitance, the load capacitance for the crystal is 7.5 pF. Note: ESS recommends 10 pF capacitors to ground on XI and XO oscillator pins.

If the crystal used is specified for a different load capacitance, it will oscillate at a slightly incorrect frequency. Usually this produces a very small, unnoticeable error in pitch. A series-type crystal oscillates about .2% faster than specified.

33.000 MHz crystals are designed to operate in either a “fundamental” or “overtone” mode. Overtone crystals are usually less expensive and can often operate at a fundamental frequency that is lower than the desired 33.000 MHz. ESS does not guarantee that all overtone crystals oscillate at 33.000 MHz without adding an LC filter to the XO output pin.

Shown in Figure 8 below is an example of an LC circuit that has a “tank” circuit tuned for a 33 MHz third overtone crystal. Resistor Ro may not be required, or may be of some low value such as 150 ohms.

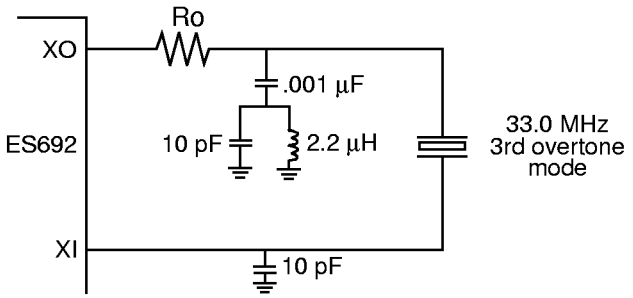


Figure 8 Using a Crystal in Overtone Mode

CONFIGURATION PROCEDURE

This section describes the steps necessary to enable the ES692 to work with an ES1xxx audio chip. The following assumptions are made:

- The control connection to the ES692 is through the MPU-401 port of the ES1xxx.
- The ES1xxx does not try to detect the ES692. If the ES692 is not present, the configuration procedure still runs unless the MCLK input to the ES692 is subject to excessive digital noise. This happens if there is a long trace between the MCLK input of the ES1xxx and the connector to an ES692 daughter card. The MCLK input has a pull-down device, but a long trace acts as an antenna when the ES692 daughter card is not plugged in. A symptom is intermittent operation of the FM synthesizer. A pull-down resistor to the MCLK pin solves this problem.
- The ES692 uses the two-wire serial link (MCLK and MSD) to gain access to the FM DAC inside the ES1xxx chip.

BIOS or DOS Startup

The following startup procedure configures the ES692 for handling DOS applications:

1. Configure and enable the MPU-401 serial interface. The desired addresses are 330h/331h. The address range can be configured for 330h/301h, 310h/311h, 320h/321h, or 330h/331h. **No interrupt is required for MIDI transmit.** It is not necessary to configure an interrupt, unless you want to run DOS applications that record MIDI input.
2. Reset the MPU-401 port. Write 0FFh to the command register (3x1) and then read back the acknowledge byte 0FEh from the data register (3x0).
3. Put the MPU-401 port in UART mode. Write 03Fh to the command register and then read back the acknowledge byte 0FEh. It is not necessary to poll or delay before sending FFh and 3Fh. It is not necessary to delay before reading the acknowledge byte.
4. Set bit 4 of mixer register 48h. If this bit is not high, the ES692 cannot use the FM DAC.
5. Program controller register 0BCh to attenuate the FM DAC output -1.5 dB relative to default. The reason is that the dynamic range of the ES692 is generally larger than FM because the number of voices is greater. Lowering the FM DAC volume by -1.5 dB reduces the chances of clipping in the analog circuits. The balance between FM and digitized audio changes slightly. After hardware reset, the content of register BCh is B6h. Write 36h to register BCh to lower the FM DAC volume by -1.5 dB.
6. Send the System Exclusive ES692 reset message. This message turns off all voices and initializes the chip. System Exclusive messages include an ID number(s) that is unique to each manufacturer. (See below.) The ESS ID number is the three byte sequence: 00h, 00h, 7Bh. The ES692 reset message is:

F0h, 00h, 00h, 7Bh	<sysex-start><id triplet>
01h	<ES692 reset command>
F7h	<end-of-sysex>

7. By default, after hardware or software rest, the ES692 audio output runs continuously. In this mode, it "owns" the DAC as long as bit 4 of mixer register 48h is set high. In Activity Detect mode, the ES692 activates MCLK only as long as it receives MIDI serial input. Specifically, if the ES692 does not detect any activity on its MSI input for 5 seconds, it turns off the MCLK output, causing the DAC to be restored to the FM synthesizer.

   It is recommended in DOS or a Windows™ DOS-box to use Activity Detect mode. This mode is activated by sending ES692 System Exclusive command 4:

F0h, 00h, 00h, 7Bh	<sysex-start><id triplet>
04h	<enable activity-detect mode>
F7h	<end-of-sysex>

   NOTE: If the ES1xxx MPU-401 port is in Smart mode rather than UART mode, MIDI data received by the chip's MSI pin echoes back out the MSO pin. In this case, MIDI data into the ES1xxx also reaches the ES692. If an external MIDI device (e.g., a keyboard) is plugged in, the device can directly drive the ES692. This is not a problem. However, some keyboards generate no-op MIDI bytes regularly at all times. These bytes are used for "active sensing," which allows MIDI devices to detect when something is unplugged from the MIDI network. Such active sensing can prevent the ES692 from releasing the DAC. Of course, if a keyboard is plugged in, it is unlikely the FM synthesizer will be used, especially in Windows.

APPLICATION SCHEMATICS

Application Schematic 1 – ES1869 with ES692

The schematic in Figure 9 shows the connections between the ES692 and the ES1869.

MIDI output from the ES1869 is directed to both the ES692 and the Joystick/MIDI connector.

MIDI output from the ES692 is connected to the MIDI input pin of the ES1869.

MIDI input from the Joystick/MIDI connector is connected to the MERGE pin of the ES692. The ES692 normally passes this signal directly through to its MSO pin except when in transmit mode.

Resistors R1 and R2 (2.2k) are for protection against static electricity.

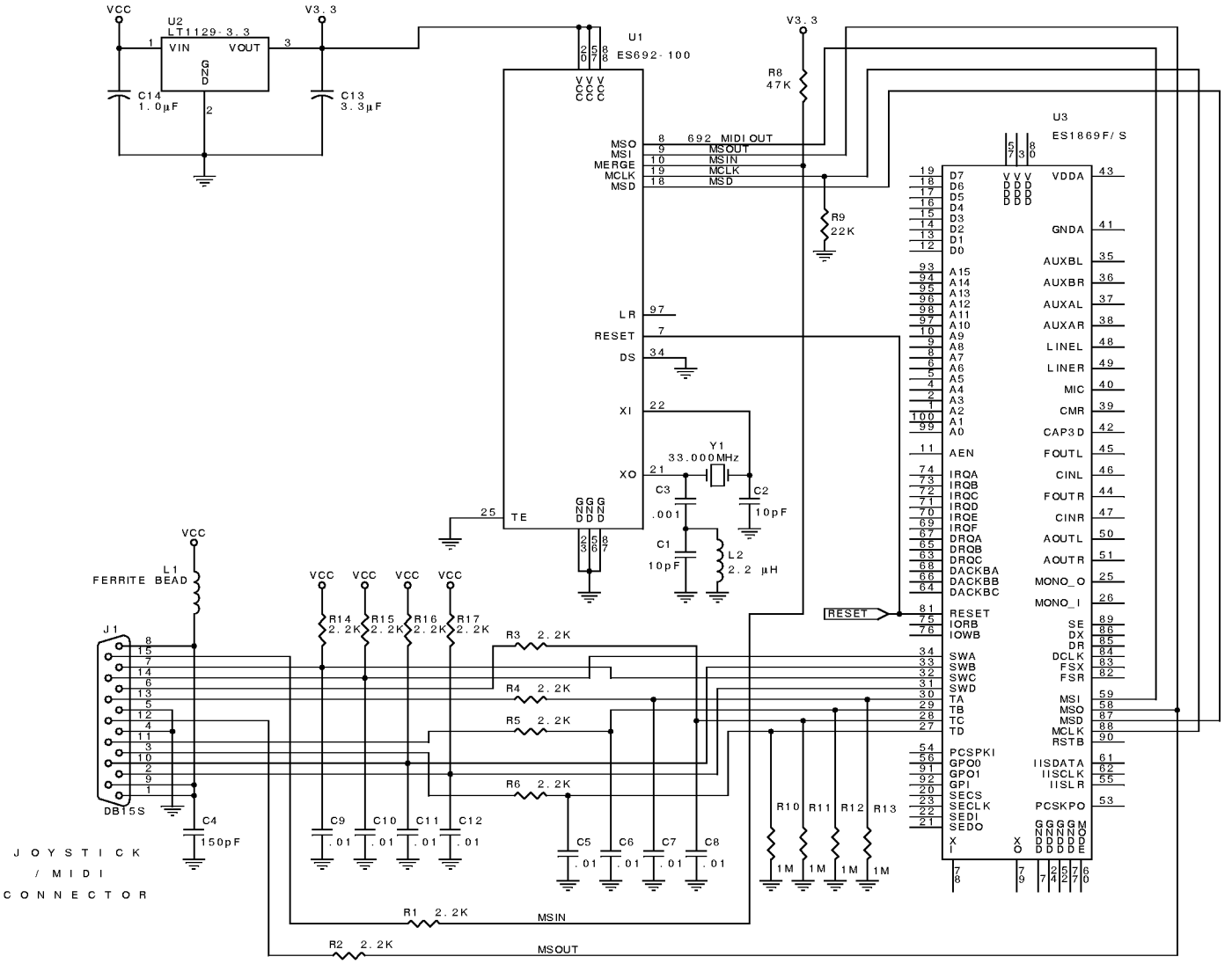


Figure 9 ES1869 with ES692

Application Schematic 2 – ES692 Daughter Card with External DAC

The schematic in Figure 10 shows an application of the ES692 as a standard wavetable daughter card, compatible with most audio cards that have a wavetable synthesizer option connector.

The header connector has analog and digital grounds. These should be kept separate on the layout.

All components should be surface mounted in order to ensure that there is enough clearance between the daughter card and the main card. Normally, the component sides of both cards face each other.

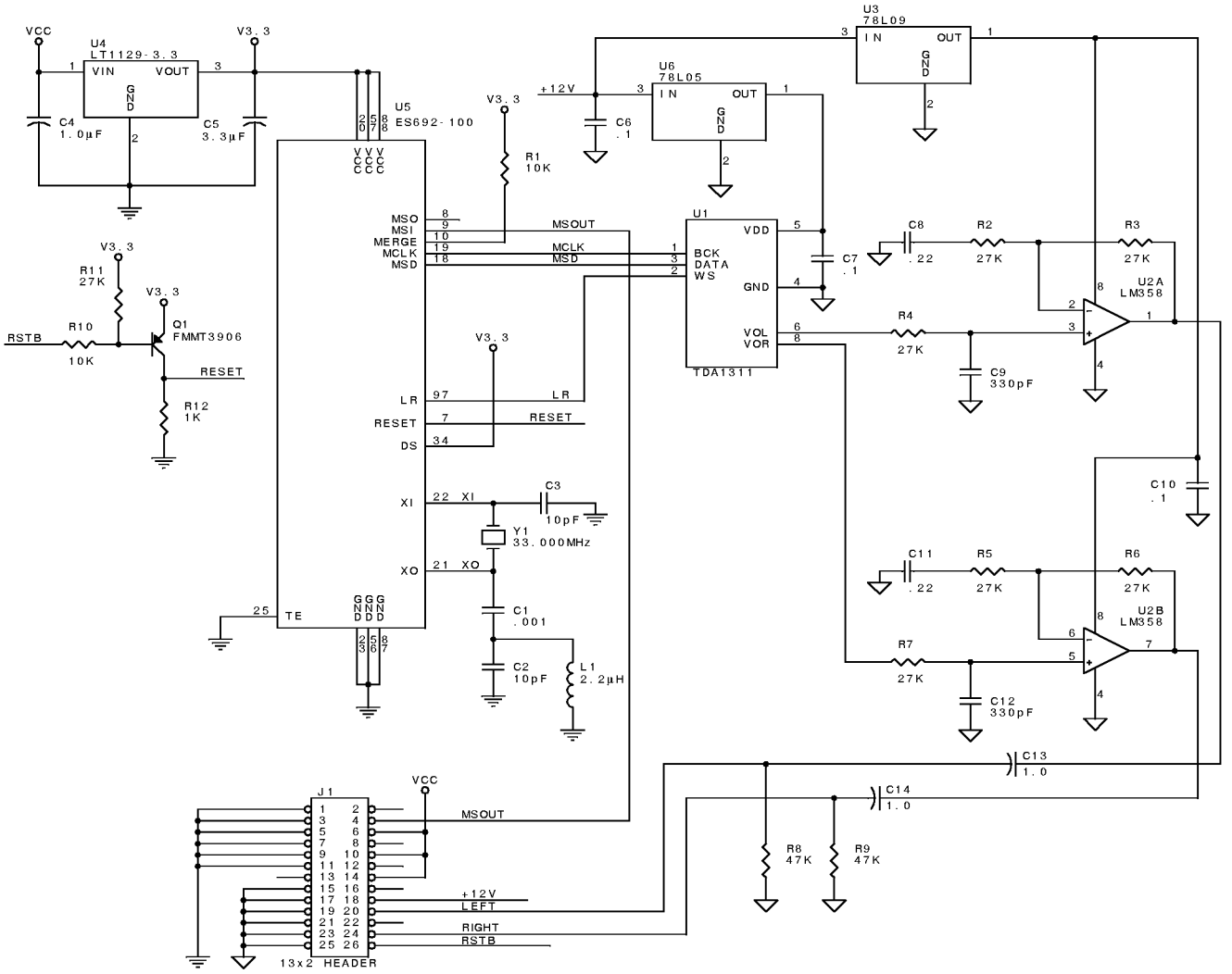


Figure 10 ES692 Daughter Card with External DAC

Application Schematic 3 – ES1868 with ES692

The schematic in Figure 11 shows the connections between the ES692 and the ES1868.

Operation of the ES1868 with the ES692 is similar to that of Application Schematic 1 – ES1869 with ES692.

For optimum performance when using the ES692 with the ES1868, use the same VCC voltage (3.3 V) for both chips. For two-way MIDI interface operation, using the same voltage (3.3 V) for both devices is a requirement.

If different VCC voltage levels are required, ESS recommends the use of voltage level shifters.

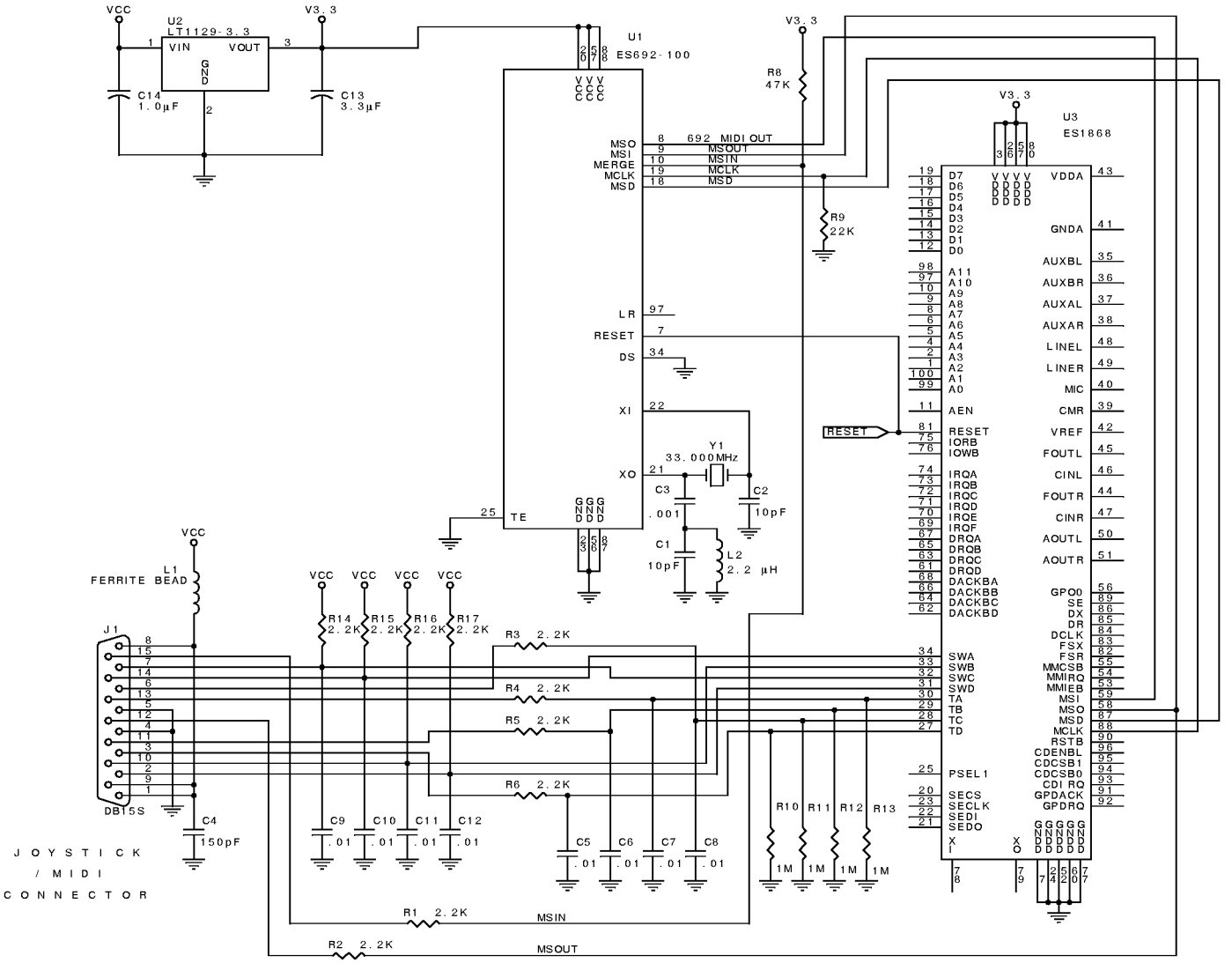


Figure 11 ES1868 with ES692

Application Schematic 4 – ES1878 with ES692

The schematic in Figure 12 shows the connections between the ES692 and the ES1878.

Operation of the ES1878 with the ES692 is similar to that of Application Schematic 1 – ES1869 with ES692.

For optimum performance when using the ES692 with the ES1878, use the same VCC voltage (3.3 V) for both chips. For two-way MIDI interface operation, using the same voltage (3.3 V) for both devices is a requirement.

If different VCC voltage levels are required, ESS recommends the use of voltage level shifters.

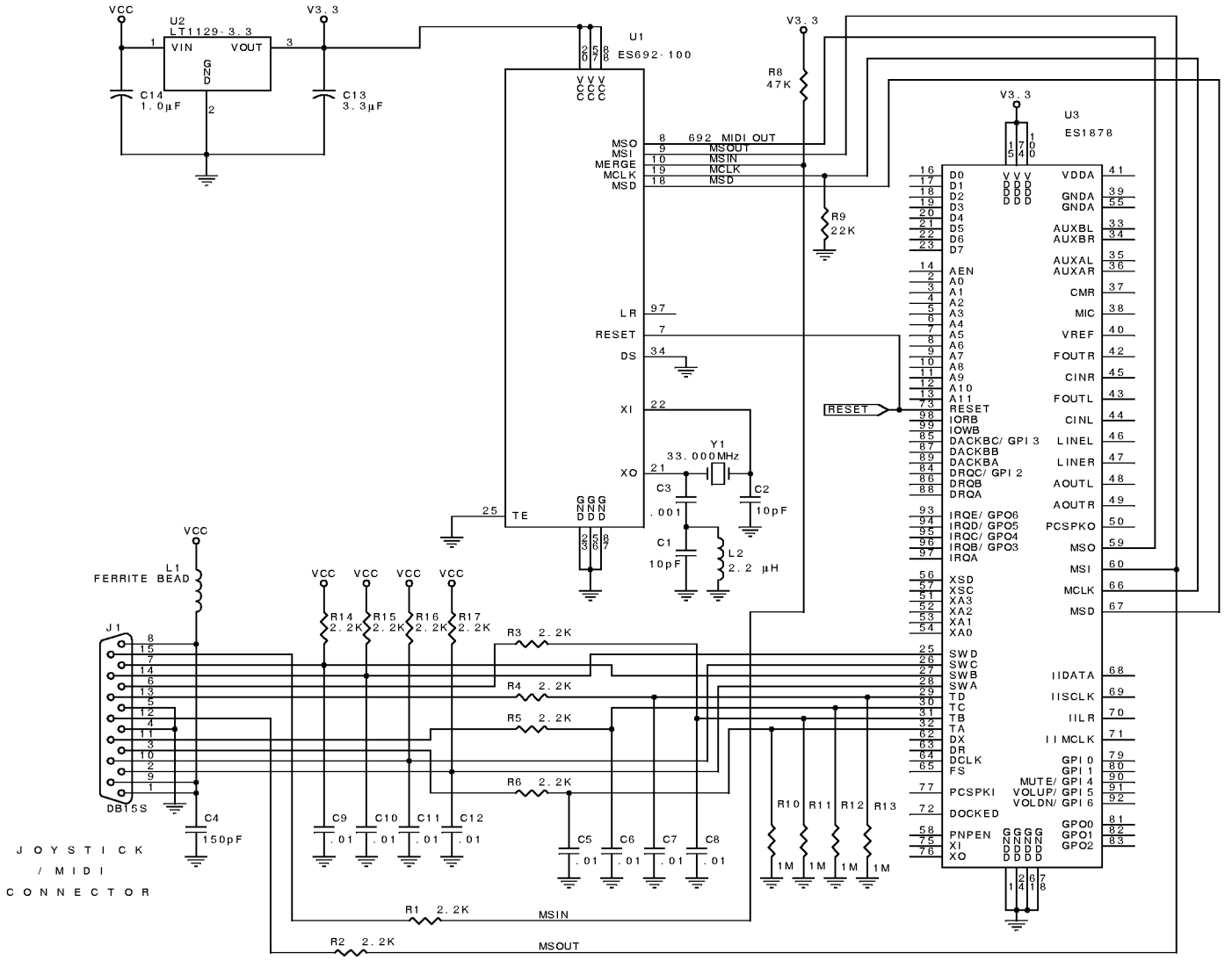


Figure 12 ES1878 with ES692

ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings

Ratings	Symbol	Value	Units
Power supply voltage	VCC	-0.3 to 7.0	V
Voltage range on any pin		-0.3 to 7.0	V
Operating temperature range	TA	0 to 70	Deg C
Storage temperature range	TSTG	-50 to 125	Deg C

WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.

Thermal Characteristics

The ES692 was designed to operate at temperatures between 0°C and +70°C.

Operating Conditions

The ES692 digital and analog characteristics operate under the following conditions:

VCC	3.0 V to 3.6 V	(3.3 volts ± 10%)
TA	25 ° C	

Table 5 ES692 Electrical Characteristics

Parameter	Symbol	Min	Typ	Max	Unit (conditions)
Operating voltage	VCC	3.0	3.3	3.6	volts
Input low voltage	VIL		0.5	0.8	volts
Input high voltage: all inputs except XI	VIH	1.2			volts
Input high voltage: XI	VIH2	2.5			volts
Output low voltage	VOL			0.4	volts (IOL = 4 mA)
Output high voltage	VOH	2.0			volts (IOH = -3 mA)
Input leakage current high	IILH			10	microamps
Input leakage current low	IILL			-10	microamps

TIMING DIAGRAM

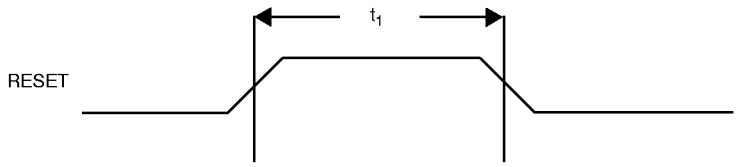


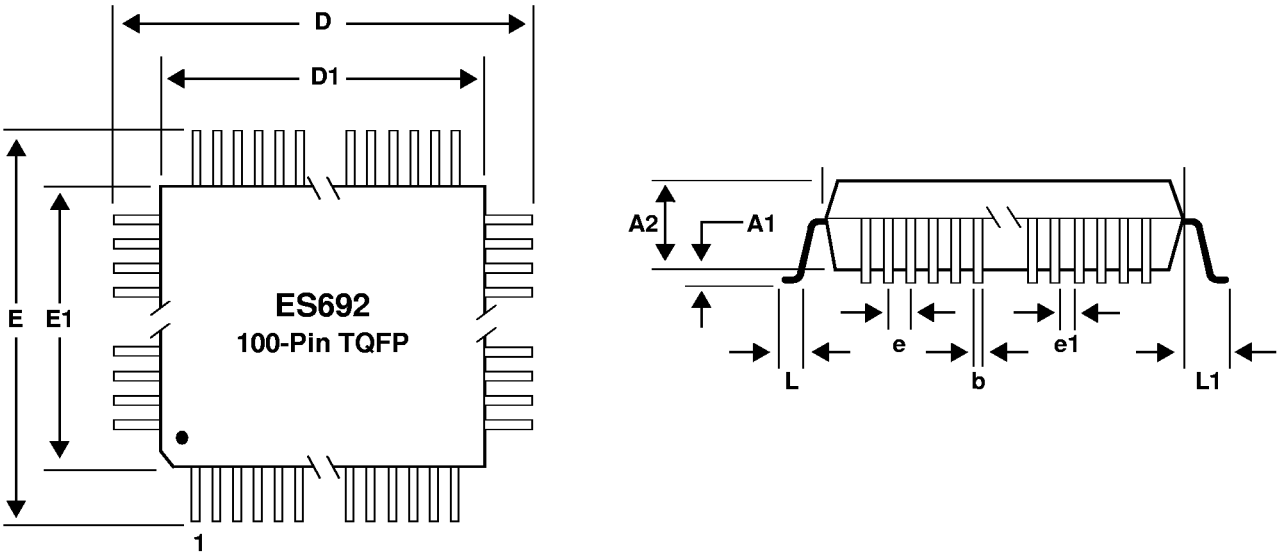
Figure 13 Reset Timing

TIMING CHARACTERISTICS

Table 6 Timing Characteristics

Symbol	Parameter	Min	Typ	Max	Units
t_1	Reset pulse width	300			ns

MECHANICAL DIMENSIONS



Symbol	Description	Millimeters		
		Min	Nom	Max
D	Lead to lead, X-axis	15.75	16.00	16.25
D1	Package's outside, X-axis	13.90	14.00	14.10
E	Lead to lead, Y-axis	15.75	16.00	16.25
E1	Package's outside, Y-axis	13.90	14.00	14.10
A1	Board standoff	0.05	0.10	0.15
A2	Package thickness	1.35	1.40	1.45
b	Lead width	0.17	0.22	0.27
e	Lead pitch	-	0.50	-
e1	Lead gap	0.24	-	-
L	Foot length	0.45	0.60	0.75
L1	Lead length	0.93	1.00	1.07
-	Foot angle	0°		7°
-	Coplanarity	-	-	0.102
-	Leads in X-axis	-	25	-
-	Leads in Y-axis	-	25	-
-	Total leads	-	100	-
-	Package type	-	TQFP	-

Figure 14 ES692 Mechanical Dimensions

APPENDIX A: SAMPLE CODES

```
/*****************************************************************************/
/*                                                                           */
/*                            ESS Technology                                 */
/*                         MPU401 Interface Library                          */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

#include "estype.h"
#include "esmpu401.h"
#include "esdelay.h"

int aiIOMPU401Choices[] = {0x330, 0x320, 0x310, 0x300, -1};


/*****************************************************************************/
/* bGetMPU401Status - Return MPU401 I/O Port Status                          */
/*****************************************************************************/
BYTE bGetMPU401Status( void )
     {
     return ( ( BYTE ) inp( _ES_MPU401_STATUS ) );
     }


/*****************************************************************************/
/* bGetMPU401RdStatus - Return MPU401 I/O Port Read Status                   */
/*****************************************************************************/
BYTE bGetMPU401RdStatus( void )
     {
     return ( ( BYTE ) ( ( ( bGetMPU401Status(  ) & _ES_MPU401_RD_MSK ) ==
               ( BYTE ) _ES_MPU401_RD_RDY ) ? 1 : 0 ) );
     }


/*****************************************************************************/
/* bGetMPU401WrStatus - Return MPU401 I/O Port Write Status                  */
/*****************************************************************************/
BYTE bGetMPU401WrStatus( void )
     {
     return ( ( BYTE ) ( ( ( bGetMPU401Status(  ) & _ES_MPU401_WR_MSK ) ==
               ( BYTE ) _ES_MPU401_WR_RDY ) ? 1 : 0 ) );
     }
```

```
/*****************************************************************************/
/* iWrMPU401Cmd - Write Command to MPU401 I/O Port                           */
/*****************************************************************************/
int iWrMPU401Cmd( BYTE _bMPU401Cmd )
    {
    int iRetVal = _ES_MPU401_ERROR,
        iTimeout = _ES_MPU401_CMD_TIMEOUT;
    while ( --iTimeout )
      if ( bGetMPU401WrStatus(  ) )
         {
         iRetVal = outp( _ES_MPU401_CMD, _bMPU401Cmd );
         break;
         }
    return ( iRetVal );
    }

/*****************************************************************************/
/* iWrMPU401Data - Write Data to MPU401 I/O Port                             */
/*****************************************************************************/
int iWrMPU401Data( BYTE _bMPU401Data )
    {
    int iRetVal = _ES_MPU401_ERROR,
        iTimeout = _ES_MPU401_WR_TIMEOUT;
    while ( --iTimeout )
          {
          if ( bGetMPU401WrStatus(  ) )
             {
             iRetVal = outp( _ES_MPU401_DATA, _bMPU401Data );
             break;
             }
          }
    return ( iRetVal );
    }
```

```
/*****************************************************************************/
/* iRdMPU401Data - Read Data from MPU401 I/O Port                            */
/*****************************************************************************/
int iRdMPU401Data( void )
    {
    int iRetVal = _ES_MPU401_ERROR,
        iTimeout = _ES_MPU401_RD_TIMEOUT;
    while ( --iTimeout )
          {
          if ( bGetMPU401RdStatus(  ) )
             {
             iRetVal = inp( _ES_MPU401_DATA );
             break;
             }
          }
    return ( iRetVal );
    }

/*****************************************************************************/
/* iResetMPU401 - Reset MPU401 I/O Port                                      */
/*****************************************************************************/
int iResetMPU401( void )
    {
    int iRetVal = _ES_MPU401_ERROR,
        iTimeout = _ES_MPU401_RST_TIMEOUT;
    while ( --iTimeout )
          {
          if ( iWrMPU401Cmd( _ES_MPU401_RESET ) != _ES_MPU401_ERROR )
             {
             if ( iRdMPU401Data(  ) == _ES_MPU401_SMART_RDY )
                {
                iRetVal = _ES_MPU401_OK;
                break;
                }
             else
                {
                }
             }
          else
             {
             }
          }
    return ( iRetVal );
    }
```

```
/*****************************************************************************/
/* iEnableSmartMode - Enable MPU401 Smart Mode                               */
/*****************************************************************************/
int iEnableSmartMode( void )
    {
    int iRetVal = _ES_MPU401_ERROR,
        iTimeout = _ES_MPU401_TIMEOUT;
    while ( --iTimeout )
          {
          iWrMPU401Cmd( _ES_MPU401_SMART_MODE );
          if ( iRdMPU401Data(  ) == _ES_MPU401_SMART_RDY )
             {
             iRetVal = _ES_MPU401_OK;
             break;
             }
          }
    return ( iRetVal );
    }

/*****************************************************************************/
/* iEnableMPU401UartMode - Enable MPU401 UART Mode                           */
/*****************************************************************************/
int iEnableUartMode( void )
    {
    int iRetVal = _ES_MPU401_ERROR,
        iTimeout = _ES_MPU401_TIMEOUT;
    while ( --iTimeout )
          {
          if ( iWrMPU401Cmd( _ES_MPU401_UART_MODE ) != _ES_MPU401_ERROR )
             if ( iRdMPU401Data(  ) == _ES_MPU401_UART_RDY )
                {
                iRetVal = _ES_MPU401_OK;
                break;
                }
             }
    return ( iRetVal );
    }
```

```
/*****************************************************************************/
/* iSendSysExMessage - Send System Exclusive Message to MPU401 I/O Port      */
/*****************************************************************************/
int iSendSysExMessage( LPBYTE _pbSysExMsg )
    {
    BYTE bIndex = 0;
    do
      {
      if ( iWrMPU401Data( _pbSysExMsg[bIndex] ) == _ES_MPU401_ERROR )
         {
         return ( _ES_MPU401_ERROR );
         }
      }
    while ( _pbSysExMsg[bIndex++] != 0xF7 );
    return ( _ES_MPU401_OK );
    }

/*****************************************************************************/
/* iReadSysExMessage - Read System Exclusive Message from MPU401 I/O Port    */
/*****************************************************************************/
int iReadSysExMessage( LPBYTE _pbSysExMsg )
    {
    int iTmpData;
    BYTE bIndex = 0;
    do
      {
      if ( ( iTmpData = iRdMPU401Data(  ) ) == _ES_MPU401_ERROR )
         {
         return ( _ES_MPU401_ERROR );
         }
      _pbSysExMsg[bIndex] = ( BYTE ) iTmpData;
      }
    while ( _pbSysExMsg[bIndex++] != 0xF7 );
    return ( _ES_MPU401_OK );
    }
```

```
/*****************************************************************************/
/* iFindMPU401IO - Find MPU401 I/O Port Base Address                         */
/*****************************************************************************/
int iFindMPU401IO( void )
    {
    BYTE bIndex;
    for ( bIndex = 0; aiIOMPU401Choices[bIndex] != -1; bIndex++ )
        {
        gwIOMPU401 = ( WORD ) aiIOMPU401Choices[bIndex];
        if ( iResetMPU401(  ) == _ES_MPU401_OK )
           {
           return ( aiIOMPU401Choices[bIndex] );
           }
        }
        return ( _ES_MPU401_ERROR );
    }
```

```
/*****************************************************************************/
/* ESMPU401.H                                                                */
/*****************************************************************************/


#define _ES_MPU401_DATA        gwIOMPU401
#define _ES_MPU401_CMD         (gwIOMPU401+1)
#define _ES_MPU401_STATUS      (gwIOMPU401+1)

#define _ES_MPU401_OK          0x00
#define _ES_MPU401_ERROR       -1


#define _ES_MPU401_RD_MSK      0x80
#define _ES_MPU401_RD_RDY      0x00

#define _ES_MPU401_WR_MSK      0x40
#define _ES_MPU401_WR_RDY      0x00

#define _ES_MPU401_RDY         0xFE

#define _ES_MPU401_SMART_RDY   0xFE
#define _ES_MPU401_UART_RDY    0xFE

#define _ES_MPU401_TIMEOUT     5000

#define _ES_MPU401_RST_TIMEOUT 10

#define _ES_MPU401_CMD_TIMEOUT 10
```

```
#define _ES_MPU401_RD_TIMEOUT  5000
#define _ES_MPU401_WR_TIMEOUT  5000

#define _ES_MPU401_RESET       0xFF
#define _ES_MPU401_SMART_MODE  0xFF
#define _ES_MPU401_UART_MODE   0x3F

extern WORD gwIOMPU401;

BYTE bGetMPU401Status( void );

BYTE bGetMPU401RdStatus( void );
BYTE bGetMPU401WrStatus( void );

int iWrMPU401Cmd( BYTE _bMPU401Cmd );

int iWrMPU401Data( BYTE _bMPU401Data );
int iRdMPU401Data( void );

int iResetMPU401( void );

int iEnableSmartMode( void );
int iEnableUartMode( void );

int iSendSysExMessage( LPBYTE _szSysExMsg );
int iReadSysExMessage( LPBYTE _szSysExMsg );

int iFindMPU401IO( void );
```

```
/*****************************************************************************/
/* ES6XX.H                                                                   */
/*****************************************************************************/
int iResetES690( void );
void vEnable689( void );
```

```
/*****************************************************************************/
/* ES6XX.C                                                                   */
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>

#include "estype.h"
#include "esmix.h"
#include "esmpu401.h"
#include "es6xx.h"
```

```
BYTE ab690SysExMessages[2][6] = { {0xF0, 0x00, 0x00, 0x7B, 0x01, 0xF7},
                                  {0xF0, 0x00, 0x00, 0x7B, 0x04, 0xF7} };
int iResetES690( void )
    {
    int iRetVal = _ES_MPU401_ERROR;
    #ifdef _DEBUG
      printf ( "iReset690\n" );
    #endif
    if ( iResetMPU401(  ) != _ES_MPU401_ERROR )
       {
       if ( iEnableUartMode(  ) != _ES_MPU401_ERROR )
          {
          if ( iSendSysExMessage( ab690SysExMessages[1] ) != _ES_MPU401_ERROR )
             {
             #ifdef _DEBUG
               printf ( "ES690 Reset\n" );
             #endif
               iRetVal = _ES_MPU401_OK;
             }
          else
             {
             #ifdef _DEBUG
               printf( "Unable to send SysEx DAC control\n" );
             #endif
             }
          }
       else
          {
          #ifdef _DEBUG
            printf( "Unable to enable UART mode\n" );
          #endif
          }
       }
    else
       {
       #ifdef _DEBUG
         printf( "Unable to reset MPU401 port\n" );
       #endif
       }
    return ( iRetVal );
    }
```

```
/*************************************************************************/
/*                                                                       */
/*************************************************************************/
void vEnable689( void )
     {
     switch ( gwIOMPU401 )
            {
            case 0x300:
              bWrMReg( 0x40, ( BYTE ) ( ( bRdMReg( 0x40 ) & 0xE7 ) | 0x00 ) );
              break;
            case 0x310:
              bWrMReg( 0x40, ( BYTE ) ( ( bRdMReg( 0x40 ) & 0xE7 ) | 0x08 ) );
              break;
            case 0x320:
              bWrMReg( 0x40, ( BYTE ) ( ( bRdMReg( 0x40 ) & 0xE7 ) | 0x10 ) );
              break;
            case 0x330:
              bWrMReg( 0x40, ( BYTE ) ( ( bRdMReg( 0x40 ) & 0xE7 ) | 0x18 ) );
              break;
            }
     bSetMRegBit( 0x48, 4 );
     iResetES690(  );
     }
```

APPENDIX B: ES692 DAUGHTER CARD SCHEMATIC

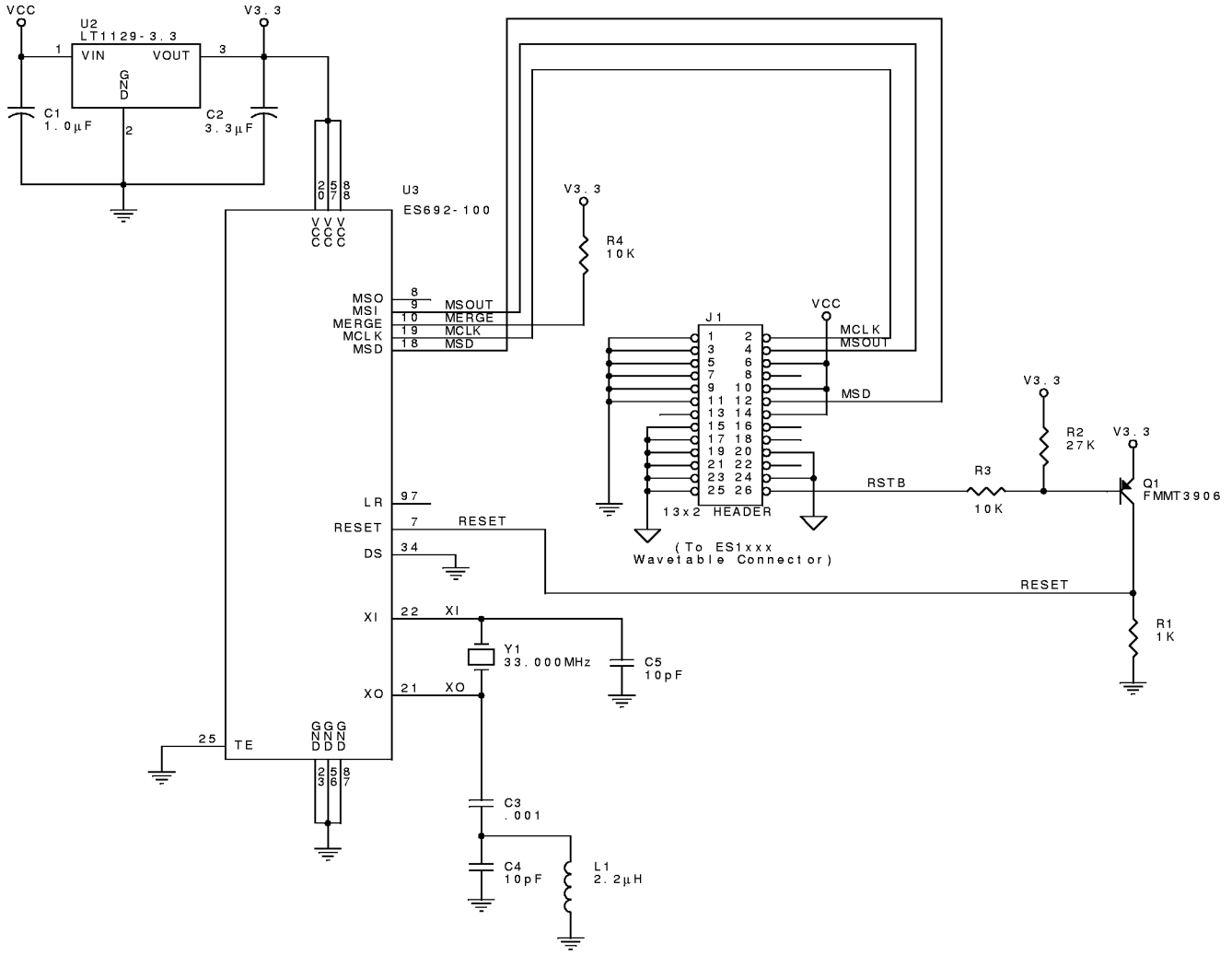


Figure 15 ES692 Daughter Card

APPENDIX C: ES692 BILL OF MATERIALS

Table 7 ES692 Bill of Materials (BOM)

Item	Quantity	Reference	Part
1	1	C1	1.0 µF Tantalum
2	1	C2	3.3 µF Tantalum
3	1	C3	0.001 µF, 0805
4	2	C4, C5	10 pF, 0805
5	1	J1	13x2 Header, female
6	1	L1	2.2 µH, 1210
7	1	Q1	FMMT3906, SOT-23
8	1	R1	1K, 0805
9	1	R2	27K, 0805
10	2	R3, R4	10K, 0805
11	1	U2	LT1129-3.3, SOT-223
12	1	U3	ES692 TQFP-100
13	1	Y1	33.000 MHz Crystal PCB Options: Epson CA-301 Metal Can Epson MA-506 Surface Mount Low Profile HC-49 or equivalent