ChipCorder®
TECHNOLOGY BY ISD

APPLICATIONS NOTE 6

A "C" Language Source Code Example to Use with the ISD33000 Products

This application note provides an example to end-users of how to use the supplied memory management algorithm to effectively manage the ISD33000 device's memory.

This applications note provides an example to end-users of how to use the supplied memory management algorithm (please see the section "Device Operation" for a more detailed discussion of this algorithm) to effectively manage the ISD33000 device's memory. This program is written in "C" language and compiled for use on the 68HC705-C8A microcontroller. Only the play and record functions are detailed here. The complete software "C" language source code for the ISD-ES301 Evaluation Board is available on ISD's website at www.isd.com.

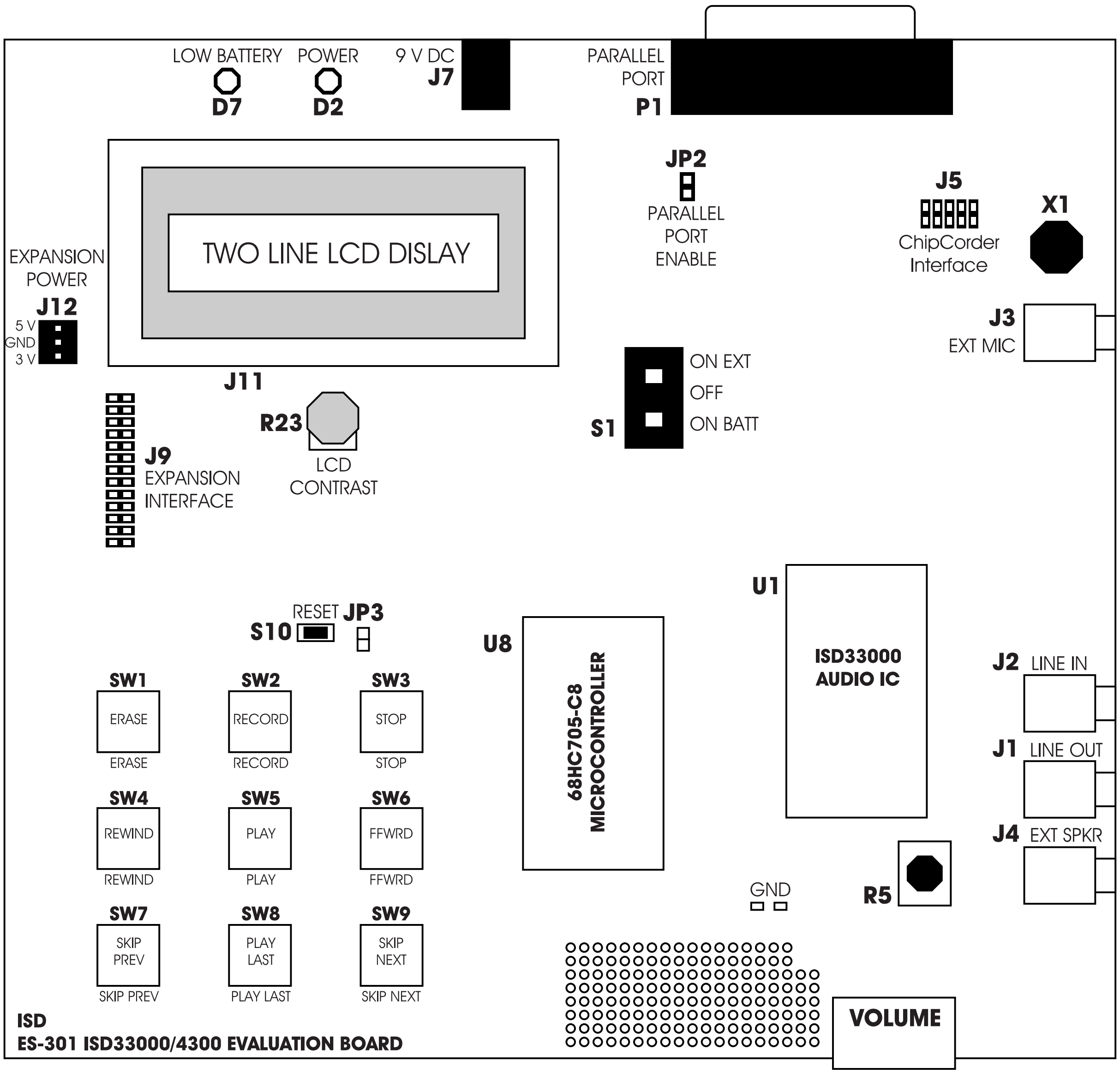
The ISD-ES301 is an evaluation and demonstration system which provides the user with the tools needed to evaluate ISD33000 analog ICs. This board can also function as a complete stand alone demonstration system. Some of the features incorporated in the software are; fast forward, rewind, skip to next message, skip to previous message, erase the current message, play, record and stop/pause a message.

The ISD-ES301 provides the following capabilities:

- Application development using the ISD33000 products
- User development of software for controlling the ISD33000 products
- Solder-in area on board for prototype installation
- Turn-key software for the Motorola microcontroller (68HC705-C8)
- Connection for SPI interface to off-board microcontroller

Figure 1, a board layout diagram, details the features of the ISD-ES301 accessible to or of importance to the user. The following Table 1 contains an alphabetical listing describing each of these features.

Figure 1: ISD-ES301 Front Panel



ISD-ES301 BOARD DESCRIPTION

The Table 1 lists all the features of the ISD-ES301 Audio Evaluation/Demo Board.

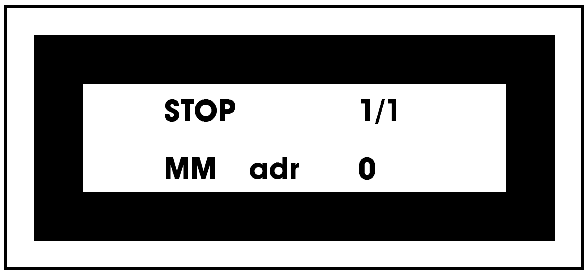
Table 1: ISD-ES301 Demo Board Feature

ID	Name	Description/Function
J1	Line Out	Direct Audio output of ISD33000 AUDOUT pin.
J2	Line In	Connection to ANA IN of ISD33000.
J3	Ext Mic	Connection for an external microphone.
J4	ext SPKR	Connection for an external speaker.
J5	chipcorder interface	Interface for external ChipCorder controller board.
J9	expansion interface	Connection for an external development board.
J11	two line lcd display	Two line LCD display for stand alone operation.
J12	expansion power	Test Points for 3 V, 5 V, and Ground
P1	parallel port	Printer cable connection to PC LPT1 port for PC operation.
R5	SCREW DRIVER ADJUST POT	Audio volume control.
R23	SCREW DRIVER ADJUST POT	LCD contrast control switch.
S1	Power switch	Power selection switch. [External Power or Internal Battery.]
S10	reset	Resets microcontroller (only).
SW1-9	keypad	Push-Button keys that make up the on board keypad.
U1	REMOVABLE ic socket	ChipCorder installation socket.
U8	microcontroller	68HC705-C8 microcontroller.
X1	Microphone	On board microphone.

MESSAGE MANAGEMENT EXAMPLE

Firmware written for the 68HC705-C8A microcontroller explains how to control the ChipCorder. The 68HC705-C8A interfaces to the ISD33000 device control I/O using the 68HC705-C8A SPI interface, interrupt, and I/O ports. A user interface consisting of nine push buttons and an LCD display is also supported by the firmware. When memory management mode is selected, the letters "MM" are displayed at the beginning of the lower line on the LCD display as shown in Figure 2.

Figure 2: Memory Management Mode



The LCD will display the following information on the first line:

- Status information such as play, record, pause, fast forward, rewind, and stop (shown)
- Number of messages in chip (1/1)
- Current message number being played (1/1)

The LCD will display the following information on the second line (messages use the entire line):

- Current Location in chip (adr 0)
- Chip overflow ("OVERFLOW")
- End of message reached ("END OF MESSAGE" or "NO MORE MESSAGES")

KEYPAD PUSH-BUTTON DESCRIPTION

The following lists the functionality of each push-button in the order they appear on the board, left to right, top to bottom.

ERASE

Erases one message at a time, starting with the last recorded message.

RECORD

In the memory management mode, pressing RECORD begins recording a new message at an address determined by the memory management system.

STOP (PAUSE)

If a message was playing, pushing this button will cause the current message to pause/stop. During record this button stops the recording process.

REWIND

In the memory management mode, REWIND decrements the row address counter 16 rows from the currently selected message.

PLAY

In memory management mode, playback begins at the "next" message in sequence. Play also resumes playing if PAUSE was previously selected.

FFWRD

In the memory management mode, FFWRD (fast forward) increments the row address counter 16 rows from the currently selected message.

SKIP PREV

In memory management mode, skips to the previous message.

PLAY LAST

Starts playing back from the beginning of the last played message, last recorded message, or last paused message.

SKIP NEXT

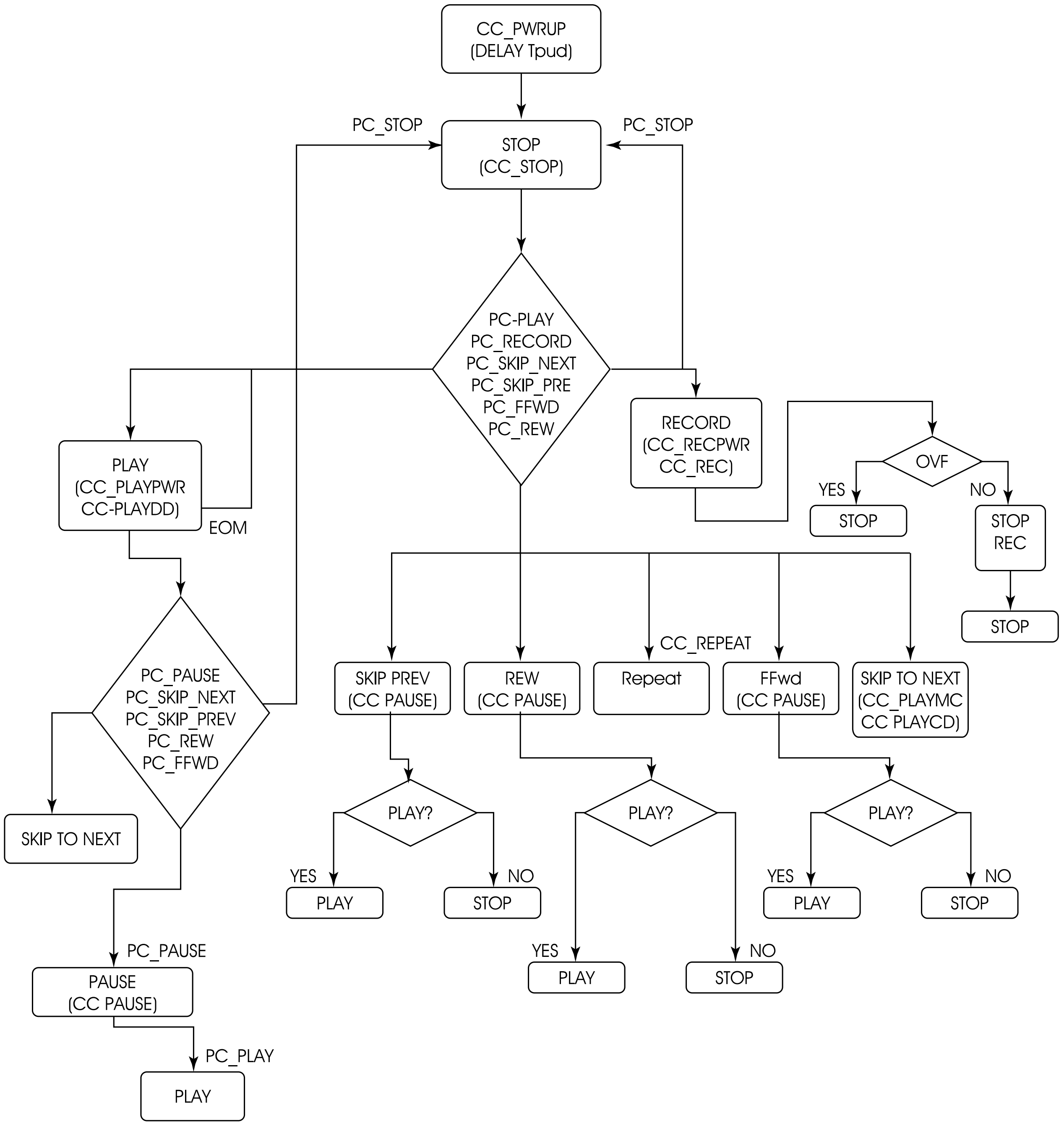
In memory management mode, skips to the next message. In manual mode, increments the row address counter 10 rows.

MEMORY MANAGEMENT MODE

This mode uses an algorithm (see "Message Management in the ISD33000 Series") which maintains a flexible message structure enabling messages to be re-recorded with different lengths while not affecting messages currently stored in the ChipCorder memory. This message structure is only maintained while the ISD-ES301 is powered up. In memory management mode the current address of the ISD33000 is updated every 16 memory rows at a time during record or playback operations. Current message number being played and the number of messages in the chip are displayed as well. Turning off power will erase the memory structure.

Figure 3 was used to write the demonstration software.

Figure 3: Push Button Control Flow Chart



```
/*----Initialize the memory address table----*/
  for ( index = 0; index < MAT_SIZE; ++index )
    MATEntry[index].All = 0x00;
   /*-----Initialize the mat structure variables-----*/
   MATStruct.EndPointer = -1;
   MATStruct.Index = 0;
/*--------------------All functions associated with Play-------------*/
void
StartPlayMessage( void )
{
   char tempStatus[ 9 ];
    /*-----if play is already running, don't re-spawn it-----*/
   if ( IsTaskRunning( RunPlayMessage ) )
      return;
   /*-----power up the isd chip-----*/
   PowerUp();
   /* ------System is in the memory management mode------ */
   {
      if ( !SystemData.NumberOfMessages )
      {
         /*----there are no messages yet in memory----*/
         SystemData.ErrorMessage ="NO MESSAGES     ";
         return;
      }
      /*----initialize reading of RAC----*/
      ReadRAC();
      /*----If the last key pressed was stop or repeat, need to
       get the BOM address,   otherwise just use the address
       in memory MemoryManagementData.Address----*/
      if ( ( KeyStatus == STOP ) || ( KeyStatus == REPEAT ) ||
      ( KeyStatus == NO_STATUS ) )
      {
         if ( SystemData.NumberOfMessages )
         {
            /*----Found messages in memory----*/
            if ( KeyStatus == REPEAT )
               /*----Set the message to be that of the last
               recorded one----*/
               SystemData.CurrentMessage = SystemData.LastRecordedMessage;
            /*----Find the address of the first block in the message----*/
            MemoryManagementData.Address =
            RetrieveBOMAddress( SystemData.CurrentMessage );
         }
         else
         {
            /*----there are no messages yet in memory----*/
            SystemData.ErrorMessage ="NO MESSAGES     ";
            return;
         }
      }
      /*----Play at the address specified by address----*/
      TransferMessageToChipCorder( CC_PLAYPWR, MemoryManagementData.Address );
      MemoryManagementData.RacCounter = 0;
      MemoryManagementData.GetAddress = TRUE;
   }
   /* ------Don't change the status if the key pressed was repeat------ */
   if ( KeyStatus != REPEAT )
      KeyStatus = PLAY;
   SpawnTask( RunPlayMessage );
   return;
} /*----StartPlay----*/
void
RunPlayMessage( void )
{
   if ( SystemMode == MEMORY_MANAGEMENT )
   {
      if ( MemoryManagementData.GetAddress )
      {
         /*----Test if time to retrieve the next address----*/
         MemoryManagementData.Address = RetrieveNextAddress( PC_PLAY );
         /*----clear get address flag----*/
         MemoryManagementData.GetAddress = FALSE;
      }/* if MemoryManagementData.GetAddress */
      if ( ReadRAC() )
      {
         /*----if RAC has changed then increment RAC counter----*/
         ++MemoryManagementData.RacCounter;
         /*----Test if the end of the block was reached----*/
         if ( MemoryManagementData.RacCounter == 1 )
            /*----Continue playing----*/
            TransferMessageToChipCorder( CC_PLAYDD, DONT_CARE );
         else if ( MemoryManagementData.RacCounter >= ROWS_PER_BLOCK )
         {
            if ( MemoryManagementData.Address <= SystemData.MaxRows )
            {
               /*----load the new block address and play at the address
               specified by address----*/
              TransferMessageToChipCorder( CC_PLAYPWR, MemoryManagementData.Address );
               /*----set GetAddress flag so next address will
               be retrieved----*/
               MemoryManagementData.GetAddress = TRUE;
               /*----Reinitialize the RAC Counter----*/
               MemoryManagementData.RacCounter = 0;
            }
          } /* else if */
      }/* if */
   }/* if MemoryManagement */
   ProcessInterrupts();
   /*----------test for end of message----------*/
   if ( SystemData.EOMFlag )
   {
      if ( InputMessage.Bit.RowPointer > ( SystemData.MaxRows - ROWS_PER_BLOCK + 1 ) )
      {
         SystemData.OVFFlag = TRUE;
         SystemData.ErrorMessage = "OVERFLOW        ";
      }
      else
         SystemData.ErrorMessage = "END OF MESSAGE  ";
      /*----Halt the playing of the message, even though it is most likely
      already stopped at this point because of the EOM flag----*/
      StopMessage();
   }
   return;
} /*----RunPlay----*/
void
StopPlayMessage( void )
{
   KillTask( RunPlayMessage );
   return;
} /*----StopPlay----*/
/*-----------------------------------------------------------------------*/
/*-----------------Retrieve Message Address for beginning of first address------------
-----*/
uint
RetrieveBOMAddress( uint msgNumber )
{
   uint msgCounter=0;
   MATStruct.Index = 0;
   do
   {
      /* ------If at the beginning of a message------ */
      if ( MATEntry[ MATStruct.Index ].Bit.MessageIndicator )
      {
         ++msgCounter;
         ++MATStruct.Index;
      }
      else
      {
         /* ------The index has surpassed the end of the messages------ */
         if ( MATStruct.Index++ >= MATStruct.EndPointer )
         {
            /*----exit the loop...at the end of the array and haven't
            found the msg----*/
            SoftwareError( NO_MESSAGES );
            return ( SystemData.MaxRows + 1 );
         }
      }
   } while ( msgCounter < msgNumber );
   /* ------Shift the address left by three because there are really
   10 bits in the message, not 7------ */
   return ( ( MATEntry[ --MATStruct.Index ].Bit.BlockPointer ) << 3 );
}     /*----RetrieveBOMAddress----*/
/*--------------------Retrieve Message Address to continue play--------*/
uint
RetrieveNextAddress( uint key )
{
   if ( MATStruct.Index < MATStruct.EndPointer )
   {
      ++MATStruct.Index;
      if ( key == PC_FFWD )
      {
         /* ------If at the beginning of the message and not at the
         end of the messages------ */
         if ( ( MATEntry[ MATStruct.Index ].Bit.MessageIndicator ) &&
         ( SystemData.CurrentMessage < SystemData.NumberOfMessages ) )
         {
            /*----Found the beginning of the next message----*/
            ++SystemData.CurrentMessage;
         }
         /*----return the next block in the MAT----*/
         return ( uint )( ( MATEntry[ MATStruct.Index ].Bit.BlockPointer ) << 3 );
      }/* if key */
      else
      {
         /*-----key must be play or repeat-----*/
         if ( !( MATEntry[ MATStruct.Index ].Bit.MessageIndicator ) )
            /*----return the next block in the MAT----*/
            return ( uint )( ( MATEntry[ MATStruct.Index ].Bit.BlockPointer ) << 3 );
      }
   }
   /*----There is no more memory to play----*/
   return ( SystemData.MaxRows + 1 );
} /*----RetrieveNextAddress----*/
/*----------------All-functions associated with record------------------*/
void
StartRecordMessage( void )
{
   /*-----if record is already running, don't re-spawn it-----*/
   if ( IsTaskRunning( RunRecordMessage ) )
      return;
   /*-----power up isd chip-----*/
   PowerUp();
   /*----The mode is memory management----*/
   {
      /*----------Find the address of the next block in the message----------*/
      MemoryManagementData.Address =
      RetrieveRecordAddress( BOM );
      if ( MemoryManagementData.Address > SystemData.MaxRows || SystemData.OVFFlag )
      {
         /*----there must be no more blocks in this message----*/
         SystemData.ErrorMessage = "NO MORE MEMORY  ";
         KeyStatus = STOP;
         return;
      }
      /*---- Record at the address specified by address ----*/
      TransferMessageToChipCorder( CC_RECPWR, MemoryManagementData.Address );
      /*----------initialize reading of RAC----------*/
      MemoryManagementData.RacCounter = 0;
      /*----------clear get address flag----------*/
      MemoryManagementData.GetAddress = TRUE;
   }/* if MemoryManagement */
   SpawnTask( RunRecordMessage );
   return;
}
void
RunRecordMessage( void )
{
   if ( SystemMode == MEMORY_MANAGEMENT )
   {
      /*----------Test if time to retrieve the next address----------*/
      if ( MemoryManagementData.GetAddress )
      {
         MemoryManagementData.Address =
         RetrieveRecordAddress( NEXT_MSG );
         /*----------clear get address flag----------*/
         MemoryManagementData.GetAddress = FALSE;
      }
      if ( ReadRAC() )
      {
         /*----------if RAC has changed then increment RAC counter----------*/
         ++MemoryManagementData.RacCounter;
         /*----------Test if the end of the block was reached----------*/
         if ( MemoryManagementData.RacCounter == 1 )
            /*----------Record at the next avail. block----------*/
            TransferMessageToChipCorder( CC_RECD, DONT_CARE );
         else if ( MemoryManagementData.RacCounter >= ROWS_PER_BLOCK )
         {
            if ( MemoryManagementData.Address < SystemData.MaxRows )
            {
               /*-------load the new block address and record at the address specified
               by address----------*/
               TransferMessageToChipCorder( CC_RECPWR, MemoryManagementData.Address );
               /*----Reset the RAC Counter----*/
               MemoryManagementData.RacCounter = 0;
           /*----------set GetAddress flag so next address will be retreived----------*/
               MemoryManagementData.GetAddress = TRUE;
            }
            else
            {
               /*-----There's no more memory to record into, we've reached the
               end of the chip-----*/
              if ( InputMessage.Bit.RowPointer >= ( SystemData.MaxRows - ROWS_PER_BLOCK
) )
                  SystemData.ErrorMessage = "OVERFLOW        ";
               else
                  SystemData.ErrorMessage = "NO MORE MEMORY  ";
               SystemData.OVFFlag = TRUE;
               StopMessage();
               return;
            }//else
         }/* else if */
      }/* if */
   } /* if */
   ProcessInterrupts();
   /*----------test for overflow----------*/
   if ( SystemData.EOMFlag )
   {
      if ( InputMessage.Bit.RowPointer >= ( SystemData.MaxRows - ROWS_PER_BLOCK ) )
         SystemData.ErrorMessage = "OVERFLOW        ";
      else
         SystemData.ErrorMessage = "NO MORE MEMORY  ";
      /*-----There's no more memory to record into, we've reached the
      end of the chip-----*/
      SystemData.OVFFlag = TRUE;
      StopMessage();
   }/* if eomflag  */
   return;
}
void
StopRecordMessage( void )
{
   /*-----power down the isd chip-----*/
   PowerDown();
   KillTask( RunRecordMessage );
   return;
}
/*-------------------------------------------------------------------*/
/*------------------------Erase a message----------------------------*/
void
EraseMessage( void )
{
   /*----Assume the message exists----*/
   uchar index, numOfBlocksToErase=1;
   uchar x;
   if ( SystemMode == MEMORY_MANAGEMENT )
   {
      if ( !SystemData.NumberOfMessages )
      {
         SystemData.ErrorMessage = "NO MESSAGES     ";
         return;
      }
      /*----Need to get the BOM address so that the MATStruct.Index will
      be set to the corresponding entry----*/
      /* ------This is a dummy call, used only to set up the MATStruct.Index
      value to the correct one for the current message------ */
      RetrieveBOMAddress( SystemData.CurrentMessage );
      /*----save the position of the first block of the message----*/
      /* The index is pointing to the next block in memory,
      not the first block of the message, so we need to subtract 1 */
      index = MATStruct.Index;
      /*----find the number of blocks of memory that need to be erased----*/
      while ( ( !MATEntry[ ++MATStruct.Index ].Bit.MessageIndicator ) &&
      ( MATStruct.Index <= MATStruct.EndPointer ) )
         ++numOfBlocksToErase;
      /*----This is the new End pointer for the MAT----*/
      MATStruct.EndPointer -= numOfBlocksToErase;
      /*----If the current message is the last one, no messages
      need to be shifted----*/
      if ( SystemData.CurrentMessage != SystemData.NumberOfMessages )
      {
         /*---- Shift all the MAT Entries up numOfBlocksToErase to
         erase the message----*/
         do
         {
            MATEntry[ index ].All = MATEntry[ index + numOfBlocksToErase ].All;
            index++;
         } while ( ( index <= MATStruct.EndPointer ) && ( ( index + numOfBlocksToErase
) < MAT_SIZE ) );
      }
      else
```

```
         --SystemData.CurrentMessage;
      /*----place 0's in the remaining MAT entries after the EndPointer----*/
      for ( x = index; x < ( index + numOfBlocksToErase ); x++ )
      {
         if ( x < MAT_SIZE )
            MATEntry[ ( x ) ].All = 0x00;
      }
      /*----Decrement the number of total messages----*/
      --SystemData.NumberOfMessages;
      if ( SystemData.NumberOfMessages &&
      ( SystemData.CurrentMessage < SystemData.LastRecordedMessage ) )
         --SystemData.LastRecordedMessage;
      KeyStatus = ERASE_MSG;
      SystemData.OVFFlag = FALSE;
      if ( SystemData.NumberOfMessages )
         /*----find the BOM of the current message----*/
         MemoryManagementData.Address =
         RetrieveBOMAddress( SystemData.CurrentMessage );
      else
         MemoryManagementData.Address = 0;
      InputMessage.Bit.RowPointer = MemoryManagementData.Address;
   }
   return;
}/* EraseMessage */
/*-------------------------------------------------------------------*/
/*-----------------Retrieve Message Address for Record---------------*/
uint
RetrieveRecordAddress( uchar bomFlag )
{
   uchar blockAddress=0;
   uchar tempIndex=0;
   if ( MATStruct.EndPointer < ( MAT_SIZE - 1 ) )
   {
      /*----If there's messages----*/
      if ( MATStruct.EndPointer >= 0 )
      {
         do
         {
            /*----Is the block Address in the MAT Table?----*/
            if ( MATEntry[ tempIndex ].Bit.BlockPointer == blockAddress )
            {
               /*----if yes, reset the index, then go look
               for the next block address----*/
               tempIndex = 0;
               ++blockAddress;
            }
            else
               /*----If the block address is not existent, make sure the
               end of the MAT has not been exceeded----*/
                  if ( ++tempIndex > MATStruct.EndPointer )
                     /*----exit the loop----*/
                     tempIndex = MAT_SIZE + 1;
         } while  ( tempIndex < MAT_SIZE );
      }
      if ( ++MATStruct.EndPointer < MAT_SIZE  )
      {
         MATEntry[ MATStruct.EndPointer ].Bit.BlockPointer = blockAddress;
         MATEntry[ MATStruct.EndPointer ].Bit.MessageIndicator = bomFlag;
         /*----Make sure that the index is pointing to the current entry----*/
         MATStruct.Index = MATStruct.EndPointer;
         return blockAddress << 3;
      }
   }
   /*----There is no more memory to record into----*/
   return ( SystemData.MaxRows + 1 );
} /*----RetrieveRecordAddress----*/
/*-------------------------------------------------------------------*/
/*---------------------------Interrupt routine-----------------------*/
#pragma TRAP_PROC SAVE_REGS
void
ExternalInterruptProc( void )
{
   /*-----tempByte, bit and i are used for the BitSwapMacro-----*/
   /*-----byteToSwap, tempInput2, tempInput1, tempByte2, and outputMessage
   are used for the TransferMessageToChipCorderMacro, -----*/
   uchar transfer, ccCommand;
   uchar bit, i;
   uint ccAddress;
   uint byteToSwap, outputMessage;
   uchar tempInput2, tempInput1;
   uchar tempByte2;
   /*-----This is to clear a dummy interrupt-----*/
   if ( SystemData.InterruptFlag )
      return;
   /*-----set the transfer flag to indicate to not send a command-----*/
   transfer = FALSE;
if ( SystemMode == MEMORY_MANAGEMENT )
   {
      switch ( KeyStatus )
      {
         case PLAY:
         case REPEAT:
             ++SystemData.RacCounter;
            /*----Test if the end of the first address was reached----*/
            if ( SystemData.RacCounter == 1 )
            {
               /*----Transfer Message to ChipCorder----*/
               (CC_PLAYDD, MemoryManagementData.Address);
            }
            /*-----test if the end of the block was reached-----*/
            else if ( SystemData.RacCounter >= ROWS_PER_BLOCK )
            {
               /*-----Means that there are more addresses in the message-----*/
               if ( MemoryManagementData.Address <= SystemData.MaxRows )
               {
                   /*----Transfer Message to ChipCorder----*/
               (CC_PLAYDD, MemoryManagementData.Address);
                  /*----set GetAddress flag so next address will
                  be retrieved----*/
                  MemoryManagementData.GetAddress = TRUE;
                  /*----Reinitialize the RAC Counter----*/
                  SystemData.RacCounter = 0;
               }
            }/* else if */
            break;
         case RECORD:
            ++SystemData.RacCounter;
            /*----Test if the end of the first address was reached----*/
            if ( SystemData.RacCounter == 1 )
            {
               /*----------Continue recording----------*/
               transfer = TRUE;
               ccCommand = CC_RECD;
               ccAddress = DONT_CARE;
            }
            /*-----test if the end of the block was reached-----*/
            else if ( SystemData.RacCounter >= ROWS_PER_BLOCK )
            {
               if ( MemoryManagementData.Address <= SystemData.MaxRows )
               {
                /*-------load the new block address and record at the address specified
                  by MemoryManagementData.Address----------*/
                  transfer = TRUE;
                  ccCommand = CC_RECPWR;
                  ccAddress = MemoryManagementData.Address;
                  /*----Reset the RAC Counter----*/
                  SystemData.RacCounter = 0;
                  /*----------set GetAddress flag so next address will be retreived---
-------*/
                  MemoryManagementData.GetAddress = TRUE;
               }
               else
               {
              if ( InputMessage.Bit.RowPointer >= ( SystemData.MaxRows - ROWS_PER_BLOCK
) )
                     SystemData.ErrorMessage = "OVERFLOW        ";
                  else
                     SystemData.ErrorMessage = "NO MORE MEMORY  ";
                  /*-----There's no more memory to record into, we've reached the
                  end of the chip-----*/
                  SystemData.OVFFlag = TRUE;
                  StopMessage();
               }/* else */
            }/* else if */
            break;
      } /* switch */
   }/* if MemoryManagement */
   else
   {
      if ( SystemData.RacCounter == 0 )
      {
         ++SystemData.RacCounter;
         if ( KeyStatus == RECORD )
         {
            /*---------Tranfer Message to ChipCorder----------*/
```

```
           cCC_RECD, DONT_CARE) ;
         }
         else if ( KeyStatus == PLAY || KeyStatus == REPEAT )
{
      TransferMessageToChipCorderMacro( ccCommand, ccAddress );
   }
}/* else */
return;
}/*-----ExternalInterruptProc-----*/
```

Figure 4: AGC Schematic

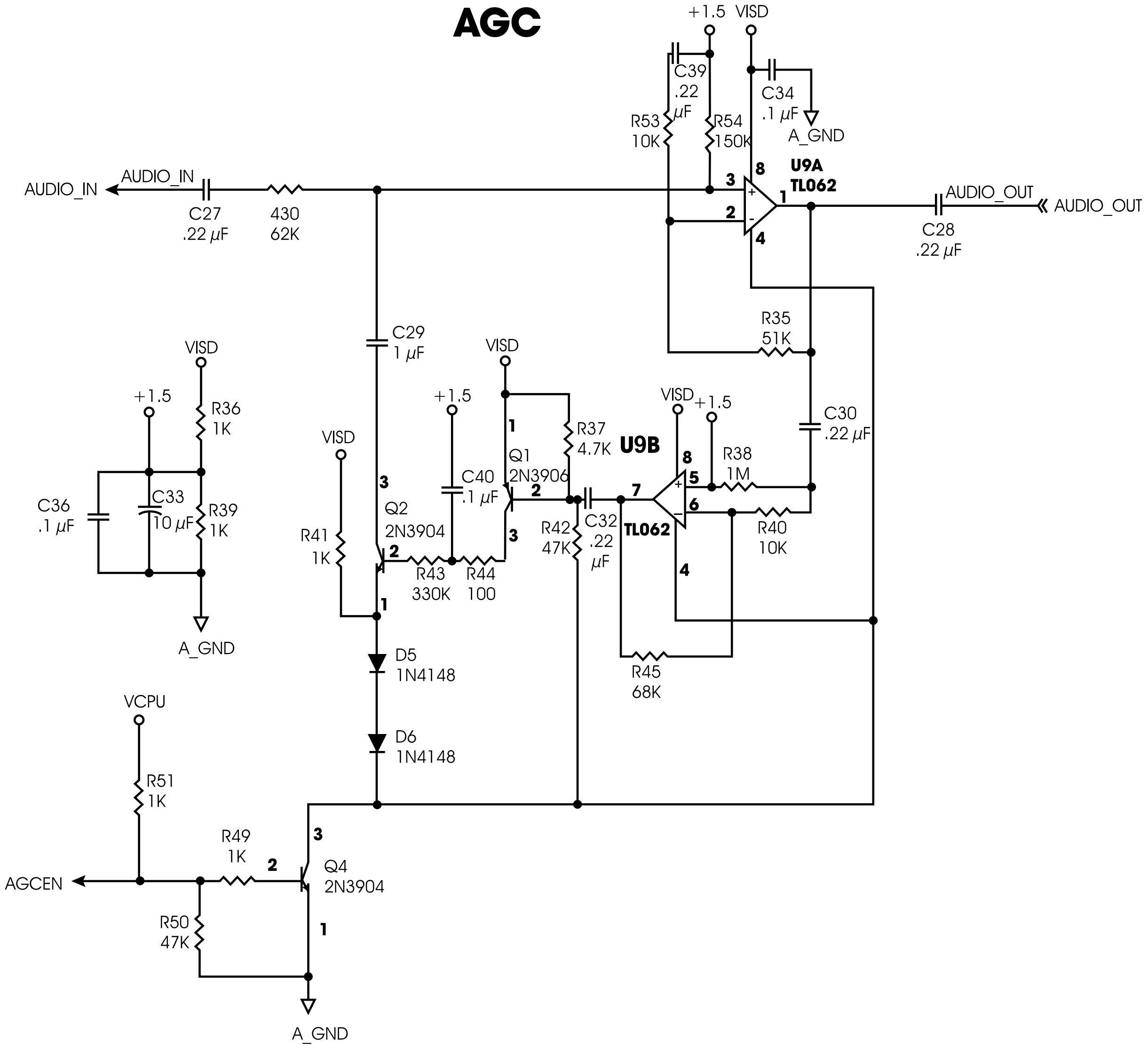


Figure 5: ISD-ES301

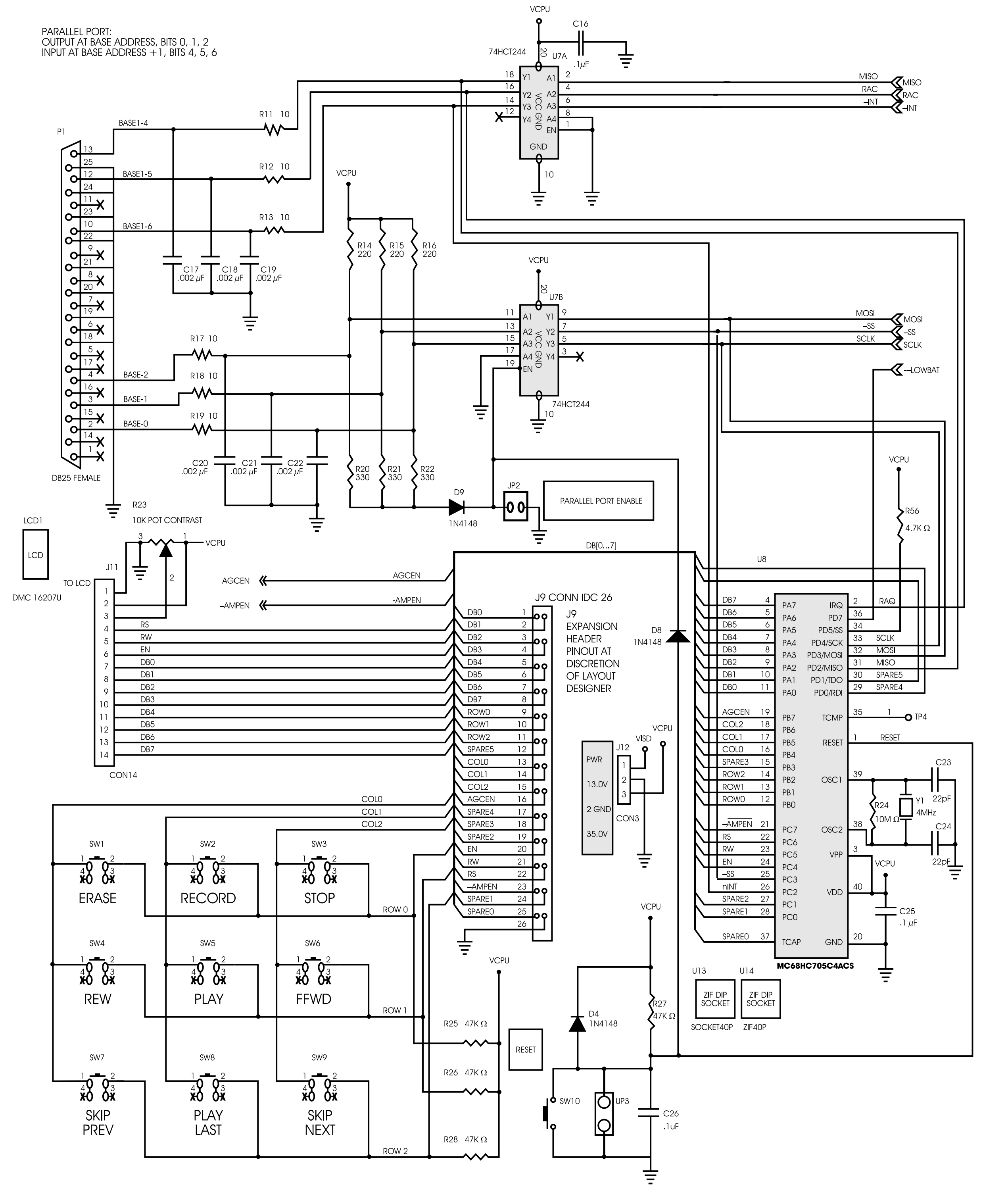
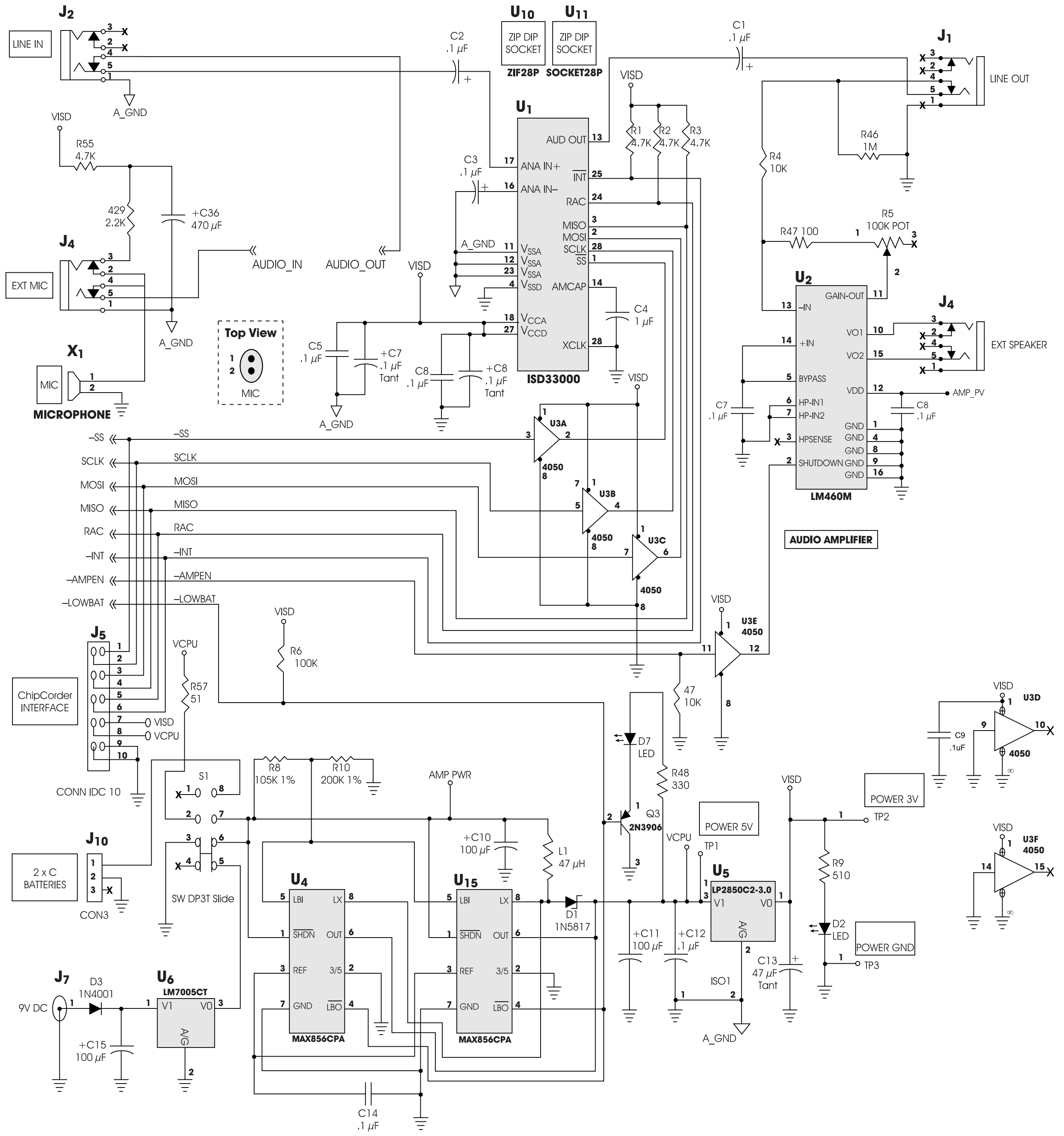


Figure 6: ISD-ES301



J2
LINE IN
A_GND
VISD
R55 4.7K
429 2.2K
+C36 470 µF
J4
EXT MIC
A_GND
X1
MIC
MICROPHONE
AUDIO_IN
AUDIO_OUT
VISD
Top View
MIC
C2 .1 µF
U10 ZIP DIP SOCKET ZIF28P
U11 ZIP DIP SOCKET SOCKET28P
U1
AUD OUT
ANA IN+
ANA IN–
INT
RAC
MISO
MOSI
SCLK
SS
AMCAP
XCLK
ISD33000
C3 .1 µF
A_GND
C4 1 µF
C5 .1 µF
+C7 .1 µF Tant
C8 .1 µF
+C8 .1 µF Tant
C1 .1 µF
J1
LINE OUT
VISD
R1 4.7K
R2 4.7K
R3 4.7K
R4 10K
R46 1M
R47 100
R5 100K POT
U2
GAIN-OUT
–IN
+IN
VO1
VO2
BYPASS
VDD
HP-IN1
HP-IN2
HPSENSE
SHUTDOWN
GND
LM460M
J4
EXT SPEAKER
AMP_PV
C7 .1 µF
C8 .1 µF
AUDIO AMPLIFIER
–SS
SCLK
MOSI
MISO
RAC
–INT
–AMPEN
–LOWBAT
U3A 4050
U3B 4050
U3C 4050
U3E 4050
U3D 4050
U3F 4050
J5
ChipCorder INTERFACE
CONN IDC 10
VISD
VCPU
R6 100K
R57 51
47 10K
C9 .1uF
J10
2 x C BATTERIES
CON3
S1
SW DP3T Slide
R8 105K 1%
R10 200K 1%
AMP PWR
+C10 100 µF
L1 47 µH
D1 1N5817
U4
LBI LX SHDN OUT REF 3/5 GND LBO
MAX856CPA
U15
MAX856CPA
D7 LED
R48 330
Q3 2N3906
POWER 5V
VCPU
TP1
U5
LP2850C2-3.0
V1 V0 A/G
+C11 100 µF
+C12 .1 µF
ISO1
A_GND
C13 47 µF Tant
POWER 3V
TP2
R9 510
D2 LED
POWER GND
TP3
J7
9V DC
D3 1N4001
+C15 100 µF
U6
LM7005CT
C14 .1 µF