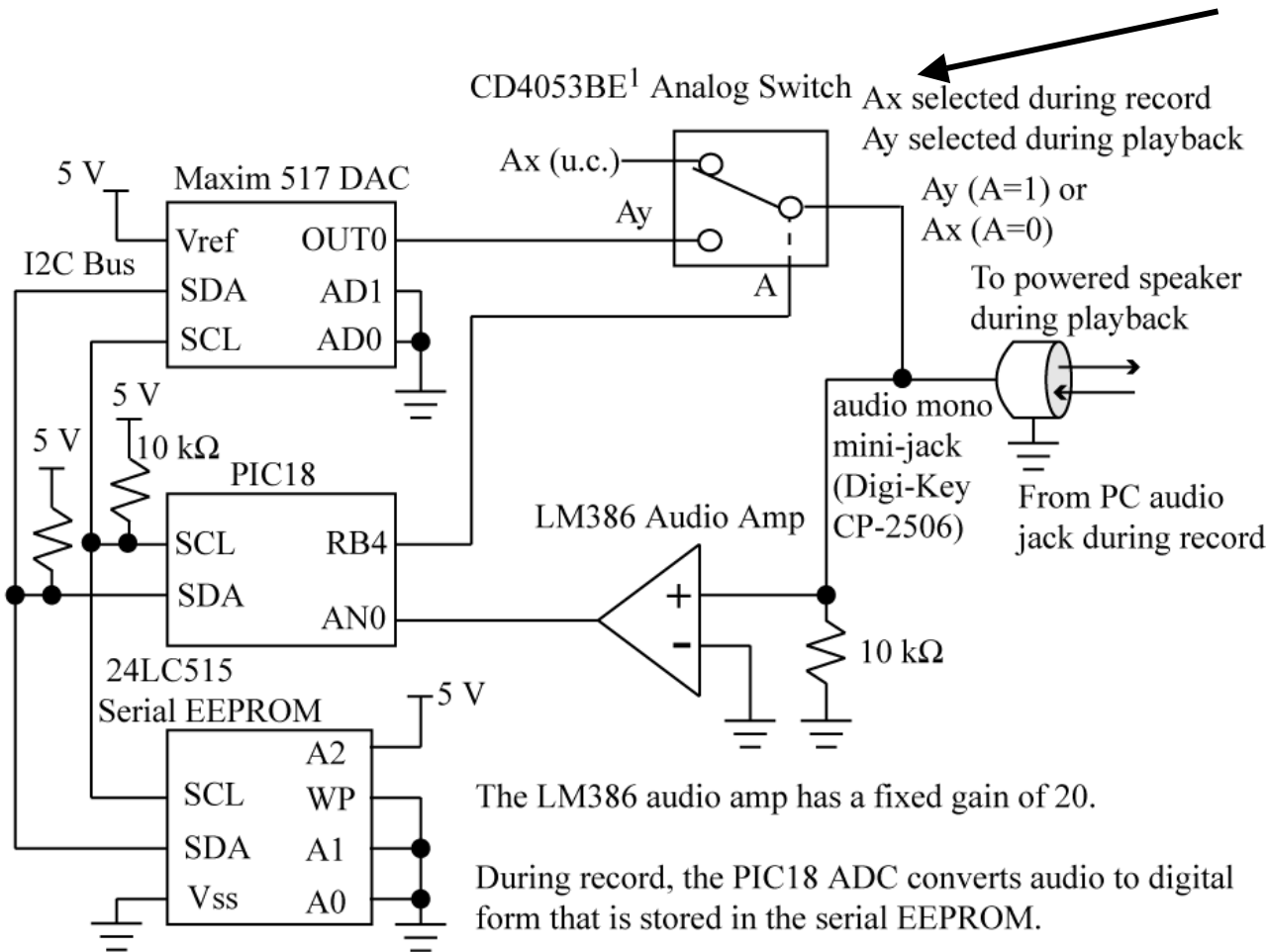


Audio Record/Playback



Used as a 'digital dip-switch'

The LM386 audio amp has a fixed gain of 20.

During record, the PIC18 ADC converts audio to digital form that is stored in the serial EEPROM.

During playback, digital audio is retrieved from the serial EEPROM and converted by the MAX 517 DAC to analog voltages that drive an external powered speaker.

¹Tie INH, VSS, VEE pins to ground.

Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

Audio Record

The file `audio.c` already provides the record and playback functions. You are asked to improve the efficiency of the playback.

Timer2 and double buffering technique (two 64-byte buffers) used to save audio to EEPROM.

All 64K bytes of EEPROM is used to save audio sample.

Sampling rate is set by Timer2.

Playback must be done at same sampling rate as record was done!!!!

How long can we record?

For voice, 8 KHz is acceptable quality

Period = 0.125 ms

Can store 64Kbytes, $64K * 0.125 \text{ ms} = 8.192 \text{ seconds}$

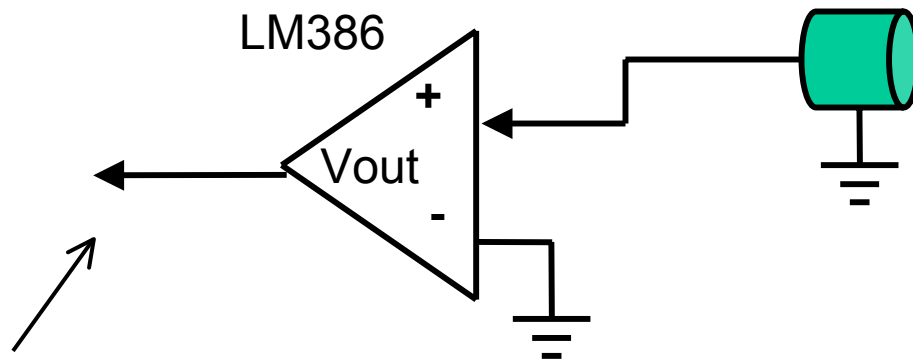
Not very long is just store raw samples. Many, many compression techniques available for voice.

However, we will just store raw 8-bit samples.

LM 386 Audio Amplifier

The LM386 is an opamp hooked up in a configuration that gives a fixed gain of 20.

Use Media player from PC to playback audio – signal out of audio jack needs amplification.



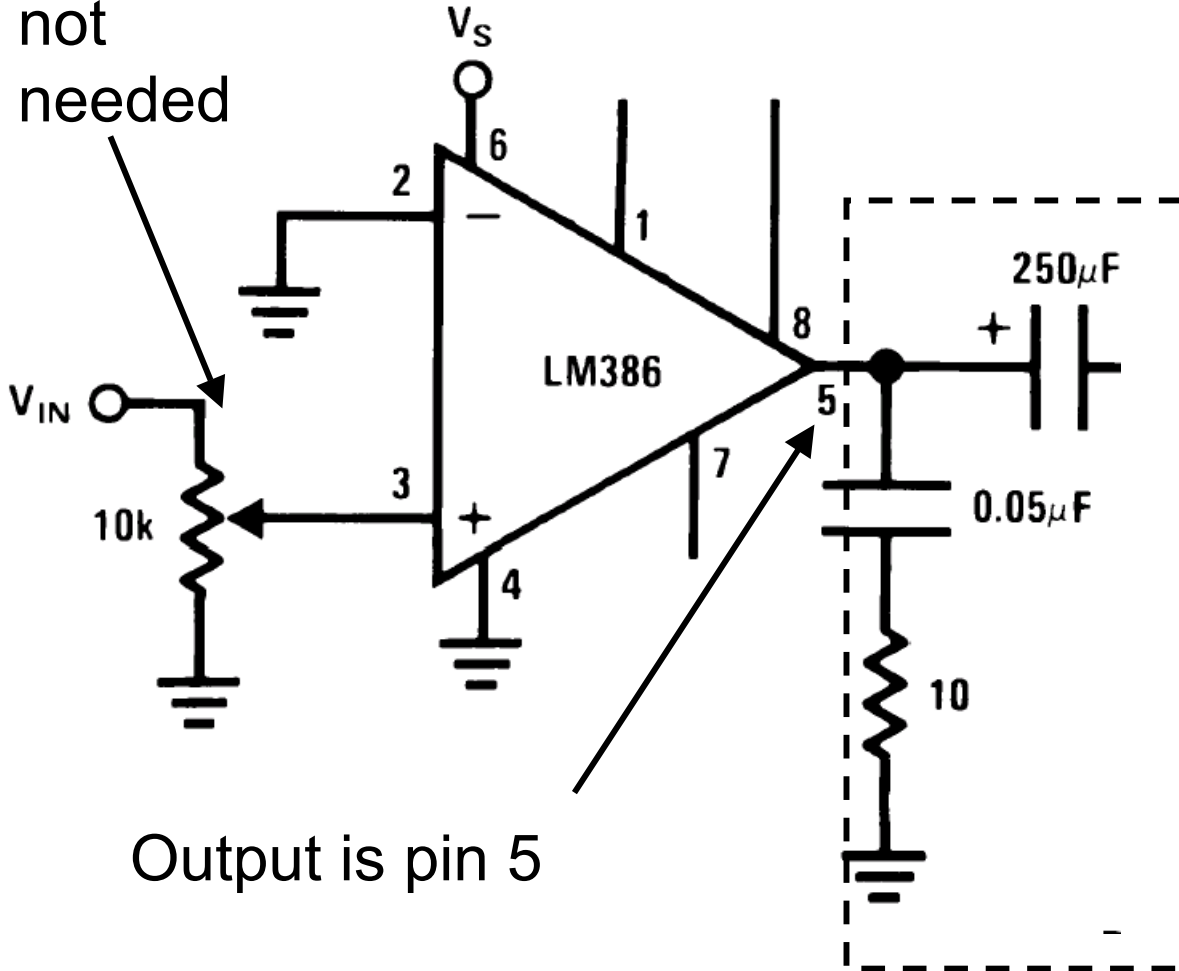
Output signal will vary about $V_{dd}/2$

Use the volume control on your PC to adjust input signal level. Use 'calibrate' mode of 'audio.c' to set this value so that you get a decent fluctuation of output values for audio input.

Amplifier with Gain = 20 Minimum Parts

LM386 –
Datasheet
view

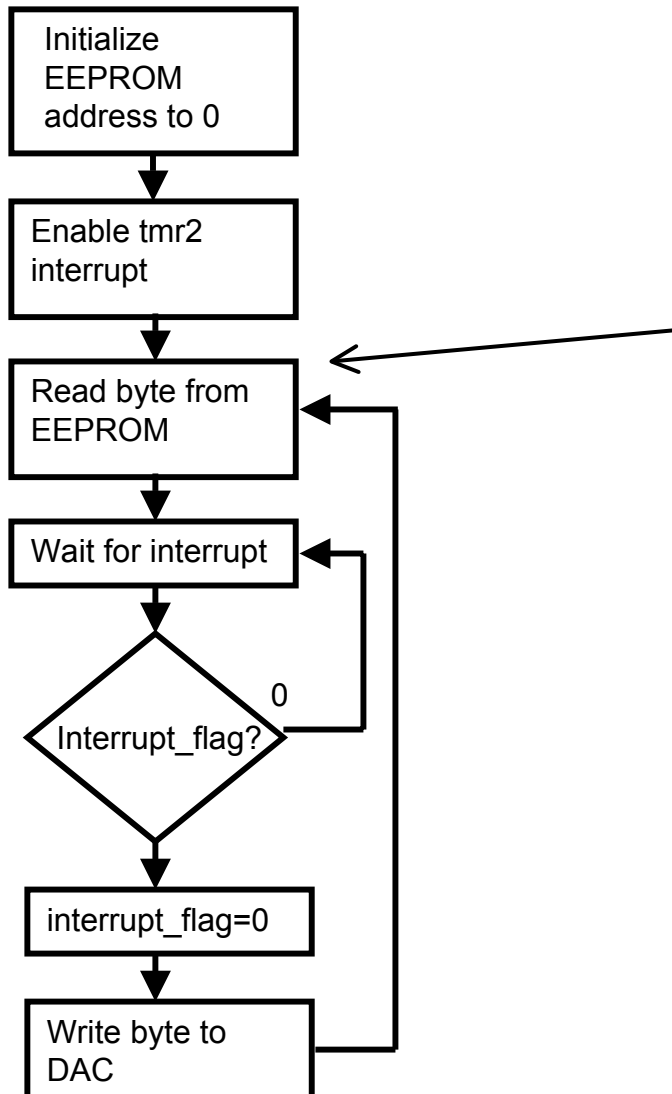
10K pot is
not
needed



Output is pin 5

Not
needed

Playback Strategy



Current address read from EEPROM.

Timer2 interrupt determines the playback rate

CAREFUL- it will be tough to achieve an 8KHz playback rate!

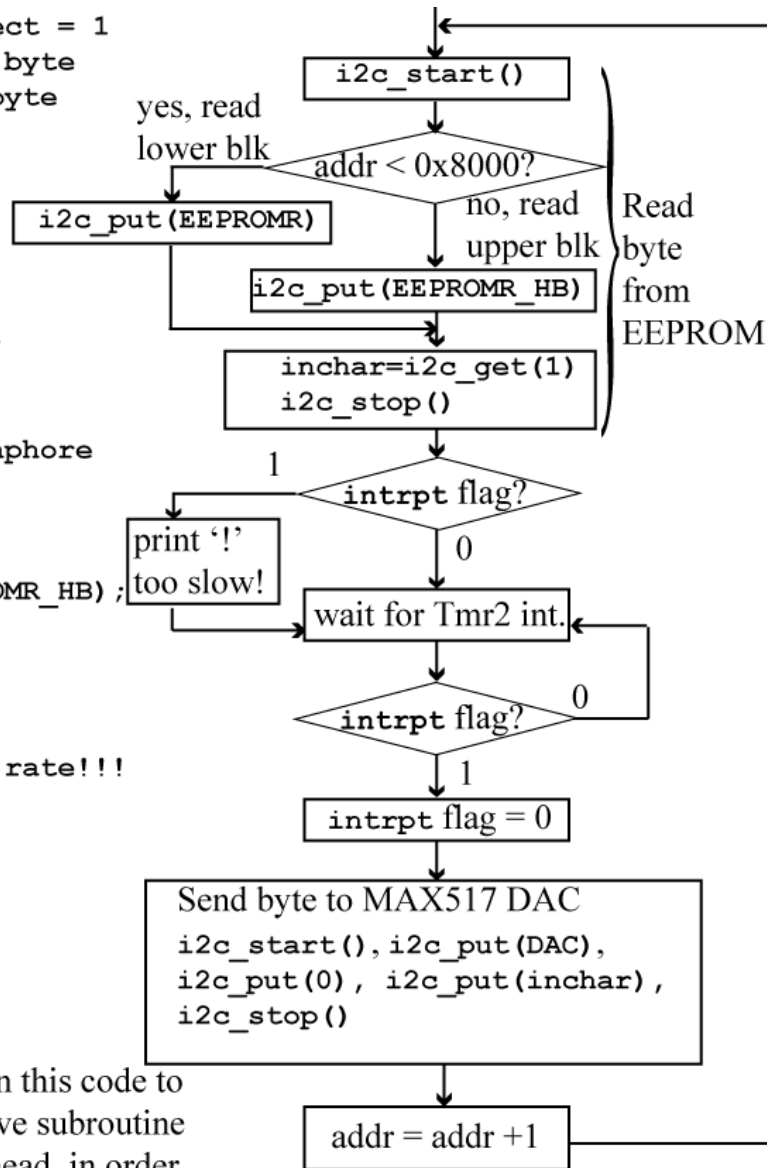
Detailed Playback Code

```

i2c_put(EEPROMW_HB); //block select = 1
i2c_put(0); // send high address byte
i2c_put(0); // send low address byte
i2c_stop(); // send stop
// initialize DAC to output zero
i2c_start(); i2c_put(DAC);
i2c_put(0x00); i2c_put(0x00);
i2c_stop();
// enable interrupts
IPEN = 0; TMR2IF = 0; TMR2IE = 1;
PEIE = 1; GIE = 1;
TMR2ON = 1 ; // start tmr2
interrupt_flag = 0; //clear semaphore
do {
    i2c_start();
    // read byte
    if (addr & 0x8000) i2c_put(EEPROMR_HB);
    else i2c_put(EEPROMR);
    inchar=i2c_get(1); //NAK
    i2c_stop();
    // if int flag is set,
    //we not keeping up with sample rate!!!
    if (interrupt_flag) putchar('!');
    // wait for interrupt
    while (!interrupt_flag);
    interrupt_flag = 0;
    i2c_start(); // send to DAC
    i2c_put(DAC);
    i2c_put(0x00);
    i2c_put(inchar);
    i2c_stop();
    addr = addr+1;
    asm("cli rwdt");
}while(1); //continual loop
}

```

flatten this code to remove subroutine overhead, in order to improve efficiency for a higher sampling rate.



Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

Playback Performance Tips

- Goal will be to reach a 8KHz playback rate
- Need to optimize the do-while(1) loop within do_playback() function
- Flatten the subroutine calls to I2C subroutines within do-while(1) loop
 - Copy code for subroutine into main loop so that don't have overhead of call/return
 - Remove idle() checks, error status (i2c_errstat keeping)
- Run the I2C bus above spec!
 - I have gotten the I2C bus to work up to about 550 KHz
 - This is cheating, and you would NOT do this in a product, but you can do it in this lab.

How do you know if code is too slow?

- When code reaches the ‘wait for interrupt’ check, if interrupt flag `ALREADY` set, then code too slow.
 - Audio quality will be degraded
 - A ‘!’ is printed to the serial port if your code is too slow
 - this helps you know if you need to speed up the code.
- The I2C bus and to some degree, the software overhead is the limiting factor of the playback
 - If used a DAC with a parallel interface could achieve a faster playback