Meeting Minutes – Group 5

Location: 0 B 6

1 March 2023, 14:30–17:30

Present

Mark Mizzi, Damjan Filipovic (Chair), Gabriel Apap

Discussion

1. Minutes from the previous meeting were read and approved.
2. Matters arising out of minutes Some issues were brought up:
   a) Some issues were brought up.
   b) Mark points out that his suggestion on avoiding C++ because of dynamic linking is incorrect as dynamic linking is not available on a bare metal platform.
   c) Gabriel points out that the statement made on having one interrupt handler per keypad button is incorrect since the keypad circuit is a diode matrix with a total of 8 usable pins.
3. Progress report from group members
   a) Gabriel researched how the LCD and the DAC functions. He has not yet looked into the amplifier circuit, but this is expected to be straightforward.
   b) Damjan has looked into lock-free queues and thinks it is possible to implement them thanks to the hardware support offered by the Cortex M4. He has not looked into the details yet due to shortage of time.
   c) Mark looked into the instruction set and found a good cheat sheet for reference. He also decided that an optimal numbering would be to give keys in the same row a sequentially ordered numbering, as then the row and column of a key can be extracted using masks and shifts. He did not have time to look at interrupts in detail.
4. Compare and contrast 2 high level designs for our system suggested by Gabriel and Mark. Based on these results, a final decision for the implementation.
   a) The problem with our previous design discussion is that we did not consider that the keypad is a diode matrix, and has to be driven using polling.
   b) Gabriel suggests setting columns high, and detecting a voltage change in any of the rows. Such a voltage change triggers an interrupt, which polls the keypad to figure out which key was pressed. Non-interrupt code processes the output queue of items generated from interrupts.
   c) Mark suggests an alternate design where timer interrupts are used to trigger polling of the keypad. This ensures that the keypad is serviced at a known frequency. Non-interrupt code processes the output queue of items as before.
   d) Mark suggests using a Gantt chart to compare the two designs.

e) Mark points out that an important design consideration is that queues must be emptied at least as fast as they are filled over some period of time otherwise the queue will overflow. Overflow is unlikely due to speed of human input.

f) Gabriel comes up with an alternate design that does not involve a queue. His idea is to process output immediately inside the polling cycle, and then return to the polling cycle. If the output code section is fast enough, any multi-key presses will be detected. This can be done inside an interrupt handler or we can just have the polling cycle as the regular code.

g) A discussion ensued where the merits of using interrupt handlers vs having polling cycle as regular code were compared. There is not much of an engineering difference between the two choices (much of the code is the same), but maybe having the code inside an interrupt handler is more future proof, since extra background tasks can then be added as regular code.

h) A decision was made to have **the polling cycle inside an interrupt handler** to allow us to add any background tasks as needed in the future. It was later decided that the initial version of the system will not work like this (see below).

i) It was pointed out that Gabriel's idea requires the use of a timer interrupt to disable the keypad-driven interrupt for some time after the last polling loop. (Since output processing happens faster than human input, not disabling the keypad interrupt for a certain amount of time would cause spurious output.)

j) Mark and Damjan point out that while Gabriel's solution is the most sophisticated (best use of resources), it is also the most complex to implement. They suggest developing the system in three separate steps:

   i. Having a system without interrupts that executes the polling cycle repeatedly in regular code. Busy waiting between polling cycles to avoid spurious output (this code will be scrapped).

   ii. Polling loop executes on timer interrupts. Output is processed during a polling loop. Regular code does nothing.

   iii. Polling loop executes on keypad-driven interrupts. Timer interrupt is used to disable keypad-driven interrupt for a certain period of time after last run of polling loop (to avoid spurious output). Regular code does nothing.

   Each of these versions of the system can reuse most of the code from the previous version, while adding more sophistication.

k) It was decided to **develop the system in these 3 steps**.

5. Discuss some core algorithms for the system.

0.1 Production of the DTMF tones in software

a) Initially the team decided to sample a single sine wave in a lookup table with 256 entries, and then set a fixed sampling period. Then to compute an approximation for a sine of period T (frequency f) at time t, we average values from the LUT according to the following formula

$$\sin(2\pi f t) \approx \frac{1}{2}\left(LUT\left\lfloor \frac{t}{T} \times 256 \right\rfloor + LUT\left\lfloor -(-\frac{t}{T} \times 256) \right\rfloor\right)$$

Note the use of the equality

$$-\left\lfloor -\frac{n}{m} \right\rfloor = \left\lceil \frac{n}{m} \right\rceil$$

to get the two entries from the LUT needed.

b) The problem with this approach is that we are given frequencies not periods. Converting these to periods will give us floating point numbers.

   Using frequencies instead of periods in the formula above does not solve this issue as then the index computation would involve 2π:

$$\sin(2\pi ft) \approx \frac{1}{2}\left(LUT\left[\frac{ft}{2\pi} \times 256\right] + LUT\left[-(-\frac{ft}{2\pi} \times 256)\right]\right)$$

c) Gabriel suggested using one sine wave's frequency f to determine the sampling frequency as:

$$f_s = \frac{f}{256}$$

   In this way, the value of this sine wave at a particular sample number i is just a simple lookup in the LUT (modulo 256).

   The value of the other sine wave (frequency f') at the same sample number i can be computed using the formula:

$$\frac{1}{2}\left(LUT\left[rem\left(\frac{f'i}{f}, 256\right)\right] + LUT\left[rem\left(-\left(-\frac{f'i}{f}\right), 256\right)\right]\right)$$

   Note that the remainder after dividing by 256 can be computed efficiently by masking with $0xff$.

d) One drawback of this approach is that the sampling frequency is not constant. However by selecting the higher frequency tone as the base, we can assure high quality sound output (sampling frequency in the range $309.5 - 418kHz$), which is below the DAC's maximum update frequency of $1MHz$ but still good quality.

e) Since the divisor is one of a set frequency of 4, the division in the formula above can actually be optimized away using magic numbers and shifts. This can be achieved using an additional LUT.

f) Gabriel pointed out that a LUT of size 256 is overkill, since the worst sampling frequency in this case is almost 10 times higher than 44.1 kHz audio. A LUT of size 32 will do. (Same formulae).

   This gives us a sampling frequency of at least $1209 \times 32 \approx 38.6$ kHz, definitely good enough.

g) Some C code was written to illustrate the idea, and the produced assembly was viewed using Compiler Explorer.

```c
int sin_lut[] = {
    2048,2447,2831,3185,3495,3750,3939,4056,
    4095,4056,3939,3750,3495,3185,2831,2447,
    2048,1648,1264,910,600,345,156,39,
    0,39,156,345,600,910,1264,1648,
};

#define SIN(baseidx, freq, basefreq) \
    ((sin_lut[(baseidx * freq / basefreq) & 0x1f] + \
      sin_lut[(-(-baseidx * freq / basefreq)) & 0x1f]) / 2)

// basefreq is the higher frequency
int sin_add(int basefreq, int freq, int idx) {
    return sin_lut[idx & 0x1f] + SIN(idx, freq, basefreq);
}
```

6. Other matters
   a) It was noted that a background task is still needed to drive the DAC, since the symbol spacing can be up to 10s (i.e. the tone can be produced for 10s).
   b) This means an output queue is still required, but only for producing the tones (LCD output and computing the tone can be handled immediately).

Actions

1. Further research into the LCD, and conversion from strings to LCD encoding.

   Assigned to: **Gabriel Apap**
   Deadline: **next meeting**

2. Research lock-free queues.

   Assigned to: **Damjan Filipovic**
   Deadline: **next meeting**

3. Research into interrupt handlers, and looking further at assembly instructions.

   Assigned to: **Mark Mizzi**
   Deadline: **next meeting**