Departament de Disseny i Programació de Sistemes Electrònics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Embedded Systems
Final exam. June 6, 2017

Duration: 3 hours. Last revision day: June 27

1 AVR and C: execution time

A common bad practice during the development of the DTMF project has been the definition of all variables as global and `volatile`. The execution time of the following code inside `TIMER0_COMPA_vect` is 8 μs and inside `if (flag){}` is 22 μs.

```
#include <avr/interrupt.h>
#include <stdbool.h>
#include <stdint.h>
#include "adc.h"
#include "tmr0.h"

#define N 200 // number of samples
#define a (int32_t)441 //coefficient scaled *256 for Fa=697 Hz
#define th 1e6

volatile int16_t s1=0,s2=0,s1_copy=0,s2_copy=0,sn=0;
volatile uint8_t xn=0,n=0;
volatile bool flag=false;
volatile uint32_t X=0;

void setup(){
  DDRD |=(1<<DDD4);//pin 4 Arduino as an output.
  DDRD |=(1<<DDD5);//pin 5 Arduino as an output.
  DDRB |=(1<<DDB5);//pin 13 Arduino as an output.
  setup_tmr0(249,8);//sampling at 8kSps
  setup_ADC(5,5,16);start_ADC();sei();
}

int main(void){
  setup();
  while(1){
    if (flag){
      PORTD |= (1<<PD5);
      X=(int32_t)s1_copy*s1_copy+(int32_t)s2_copy*s2_copy-(a*s1_copy>>8)*s2_copy;
      if (X>th){PORTB |= (1<<PB5);}
      else{PORTB &= ~(1<<PB5);}
      flag=false;
      PORTD &= ~(1<<PD5);
    }
  }
  return 0;
}

ISR(TIMER0_COMPA_vect){
  PORTD |= (1<<PD4);
  xn=read8_ADC();start_ADC();//reads 8 bits and starts a new ADC conversion
  sn=xn+(a*s1>>8)-s2;s2=s1;s1=sn;n++;//Goertzel algorithm
  if (n==N){s1_copy=s1;s2_copy=s2;s1=0;s2=0;n=0;flag=true;}
  PORTD &= ~(1<<PD4);
}
```

1. Try to reduce the execution time by just changing the type-qualifiers (`const`, `volatile`), the storage class (`static`) and the place of definition of each variable. Using this strategy the execution time has been reduced inside `TIMER0_COMPA_vect` by 15% and inside `if (flag){}` by 7.5%.

2. Initialize at the right value only the variables that need it.

2 AVR and C: CPU usage

Consider an implementation of the DTMF project in `C` using a sampling frequency $F_s = 10\,\text{kHz}$ and windows made of $N = 100$ samples. The execution time of the function `compute_sample()` is $40\,\mu\text{s}$. The execution time of the function `compute_power()` is $420\,\mu\text{s}$ when interruptions are disabled. In the following consider all other execution times zero.

```
// include, define...
#define N 100
// variable definition, functions...

int main(void){
    // variable definition, setup...
    while(1){
    if (flag){
      PORTD |= (1<<PD5);
      compute_power();// power of 8 frequencies
      flag=false;
      PORTD &= ~(1<<PD5);
    }
  }
  return 0;
}

ISR(TIMER0_COMPA_vect){
  //variable definition...
  PORTD |= (1<<PD4);
  xn=read8_ADC();start_ADC();
  compute_sample();// Goertzel algorithm for 8 frequencies
  n++;
  if (n==N){
  // copy, initialize...
  flag=true;
  }
  PORTD &= ~(1<<PD4);
}
```

1. Compute the CPU usage (in %) due to the function `compute_sample()`.

2. Draw the signals `PORTD4` and `PORTD5` seen on an oscilloscope triggered with the rising edge of `PORTD5` with the base time at $100\,\mu\text{s}$.

3. Compute the overall CPU usage (in %) due to `compute_sample()`and `compute_power()` .

3 FPGA and VHDL: synchronizing clock enable with data

A signal called `clk_en_in` is high the first rising edge of `clk` after `data_in` is ready and is low the next rising edge. We want to design an entity in order to update `data_out` from `data_in` every two `clk_en_in`. In addition, a signal called `clk_en_out` must be high the first rising edge of `clk` after `data_out` is ready and must be low the next rising edge. The following `VHDL` code tries to do that.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity se_exam_2017 is
  port (clk        : in  std_logic;--100kHz
        clk_en_in  : in  std_logic;-- 10kHz
        data_in    : in  std_logic;
        clk_en_out: out std_logic;
        data_out   : out std_logic
        );
end entity;

architecture arch_1 of se_exam_2017 is
  signal n : unsigned(7 downto 0):= to_unsigned(0,8);
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if clk_en_in = '1' then
        if n = 1 then
          clk_en_out <='1';
          data_out <= data_in;
          n <= to_unsigned(0,8);
        else
          clk_en_out <='0';
          n <= n+1;
        end if;
      end if;
    end if;
  end process;
end architecture;
```

1. Unfortunately, `clk_en_out` is not well generated. Draw the actual digital waveform (`clk`, `clk_en_in` and `clk_en_out`).

2. Modify the code and draw the right digital waveform.

4 ADC: changing the analog reference

During the project course we have used different platforms to detect a DTMF signaling. When working with Arduino we have taken the eight most significant bits of the 10-bit ADC of the AVR microcontroller with a range that goes from ground to 5 V. When working with DE0-Nano we have taken the eight most significant bits of the 12-bit ADC, external to the FPGA, with a range that goes from ground to 3.3 V.

When testing one of the DTMF signal test with Arduino we used $th_{AVR} = 10^6$ (i.e. what during the project we called the threshold, a level with which to compare the power of each one of the DFT components computed by the Goertzel algorithm).

1. Find out the value of the threshold, th_{FPGA}, that should be used with the FPGA to obtain the same results with exactly the same DTMF signal test. Consider that in all platforms we use the same DTMF signal test, with a peak-to-peak amplitude of $3\,\mathrm{V_{pp}}$.

5 Serial communication

1. Samples of 8 bit, sampled at a sampling frequency F_s, are send from an Arduino to the Raspberry Pi (RasPi) through a 57 600 bps serial connection (USART to USB bridge) using 1-start-bit, 1-parity-bit and 1-stop-bit. What is the maximum F_s you could use?

2. Consider an Arduino trying to send samples sampled at $F_s = 8\,\mathrm{kHz}$ through a 57 600 bps serial connection. What happens? Quantify the effect.

3. The measured maximum, mean and minimum time used by the RasPi B + to read L bytes, for any L between 1 and 1024, stored in the system buffer are approximately 500 μs, 450 μs and 300 μs respectively. Now consider the following code (in the next questions ignore the execution time of the functions `print` and `time.time()`).

```
import time
import serial
# parameters
L=...
# open, wait and clean serial port
ser = serial.Serial('/dev/ttyACM0', baudrate , timeout=1)
time.sleep(2)
ser.flushInput()
while 1:
        t0=time.time()
        x=ser.read(L)
        t1=time.time()
        print t1-t0
```

   a) If, as in the previous question, Arduino is sending samples at $F_s = 8\,\mathrm{kHz}$ through a 57 600 bps serial connection to the RasPi, which is the mean of the printed value $t_1 - t_0$? Let $L = 1$.

   b) Now let $L = 100$.

4. Now we add some code to compute something (e.g. an implementation of the Goertzel algorithm) after reading the serial port.

```
while 1:
        t0=time.time()
        x=ser.read(197)
        compute_something(x)
        t1=time.time()
        print t1-t0
```

   a) Consider an Arduino sending samples at $F_s = 8\,\mathrm{kHz}$ through a 115 200 bps serial connection to the RasPi. The mean of the printed value $t_1 - t_0 = 24.625\,\mathrm{ms}$. Can you

determine the execution time of the function `compute_something()`? Can you limit its value?

b) Repeat the previous question considering that the mean of the printed value $t_1 - t_0 = 28\,\text{ms}$. What percentage of the samples are lost?

5. If the operating system of the RasPi makes us wait 250 ms, what is the maximum sampling frequency F_s at which samples could arrive without losing them, considering that the size of the system buffer is 4095 bytes? Consider the best case, i.e. when the buffer is empty.

6 Power consumption

Consider that the consumption of a RasPi with some peripherals is 5 W. It is powered by a power supply system made by 2 NiMH AA batteries, each one with a nominal voltage of 1.2 V and a capacity $C_0 = 2200\,\text{mAh}$ (for any current discharge, to make it easy), followed by an appropriate DC-DC converter (with a 92% efficiency).

1. Compute the running time of the RasPi before the batteries are empty.

7 DTMF project course

1. You want to modify the eight standard frequencies used in the DTMF signaling with a sampling frequency $F_s = 8\,\text{kHz}$. Your goal is that the DFT components computed by the Goertzel algorithm be the same frequencies used in the DTMF signaling. How will you proceed?

2. You want to design your own DTMF signaling (taking windows of 25 ms) increasing the eight frequencies used by the standard to the maximum allowed by your chosen platform. This way, instead of 4_low_freq x 4_high_freq =16 DTMF signals when using 8 different frequencies, you will have 4_low_freq x 5_high_freq =20 DTMF signals when using 9 different frequencies, 5_low_freq x 5_high_freq =25 DTMF signals when using 10 different frequencies, and so on. You are in the stage of choosing your development platform. Based on the following results, obtained during this DTMF project course, make your decision: Arduino/ATmega328P: 45% CPU usage. DE0-Nano/Altera Cyclone IV EP4CE22F17C6N FPGA: 1245/22300 logic elements; 16/132 multipliers. RasPi B+: 20 ms computation time; consider zero reading time from the system serial buffer. RasPi 3 B+: 4 ms computation time; consider zero reading time from the system serial buffer.

3. You have to build your own DTMF signaling in an FPGA. As we have seen in classroom, the three 16-bit multiplications involved in the computation of the power of one frequency use six embedded 9-bit multipliers. This power computation lasts one clock of the 256 available between power computations. We have also seen that the two embedded 9-bit multipliers needed in each one of the 16-bit multiplications can be shared at the cost of incrementing each power computation time: now each power computation lasts five clocks of the 256 available between power computations. In an FPGA with four embedded 9-bit multipliers, how many frequency power computations could we make?

4. Given the poor computation time results running a code written in `python` over the old RasPi B+, propose the use of other languages to improve these results.