Mesure de la période d'horloge du bus SPI.

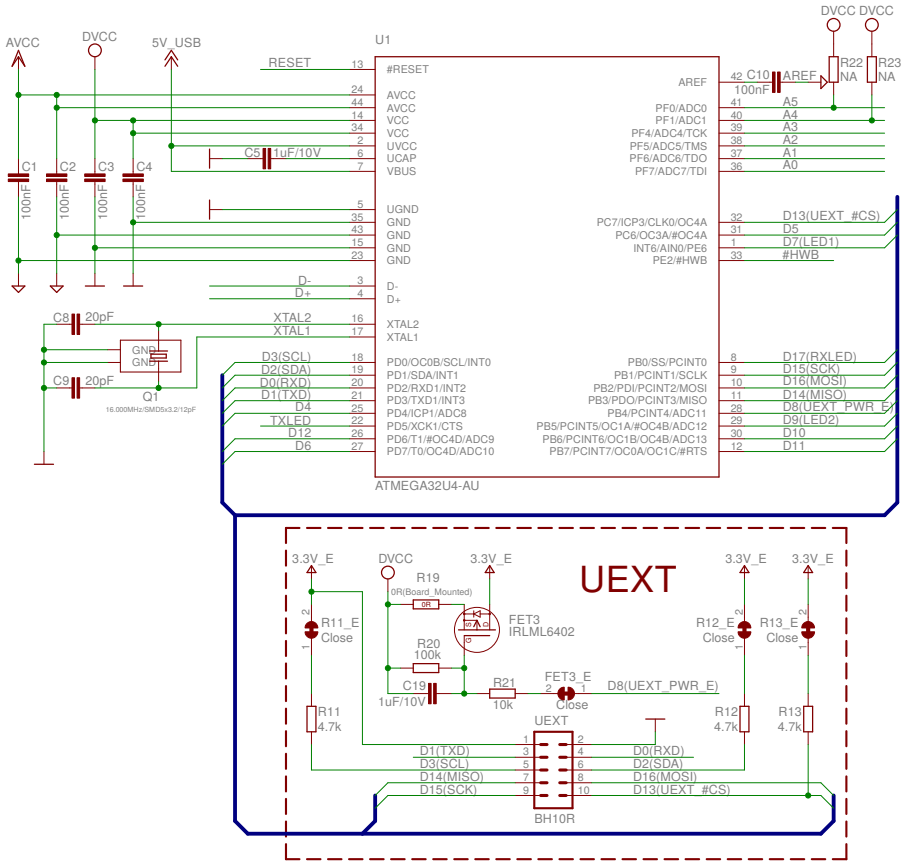
É. Carry, J.-M Friedt, 11 mars 2020

Connexion internet interdite, téléphones portables interdits, communications interdites, réflexion autorisée.
Supports de cours à `172.20.133.70`

Nous nous proposons de générer un signal d'horloge pour cadencer le bus SPI et de vérifier sa période par la mesure sur un timer.

Nous proposons trois fonctions `.c` qui fournissent chacune certaines fonctionnalités nécessaires à un programme fonctionnel. En particulier, `input_capture.c` stocke dans une variable `res` la valeur du compteur lors du déclenchement d'un événement.

1. Identifier sur le schéma de l'Olimexino32U4 les broches sur lesquelles l'horloge SPI et l'entrée timer sont connectées. Quelles sont-elles ? brancher un câble entre ces deux broches. On notera que sur le connecteur UEXT, la broche connectée à une empreinte carrée est la broche 1, et la nomenclature suit une ligne longue impaire et une ligne longue paire.



2. Proposer un `Makefile` qui automatise la compilation et l'édition de lien des programmes pour fournir un exécutable fonctionnel. On s'interdira évidemment de fusionner les trois programmes en un unique code source mais conserverons la structure de compilation séparée. Flasher le microcontrôleur.
3. Observer à l'oscilloscope la période du signal d'horloge de SPI : quelle est-elle. Est-elle en accord avec vos attentes à la lecture des programmes fournis.
4. L'interruption `input_capture.c` mémorise le compteur au moment d'une transition sur la broche associée. Modifier `input_capture.c` afin de mémoriser les transitions consécutives de l'horloge lors de l'émission d'un octet sur bus SPI. Combien de mot mémorisons nous ? sur combien de bits est stocké chaque mot ?
5. Un compteur aura pu servir à mémoriser la case mémoire contenant la dernière mesure : exporter cette variable vers `main.c` (comment faire ?) et tester cette variable pour afficher, dans la boucle infinie de `main.c`, les mesures obtenues par le timer.
6. Démontrer l'affichage de ces mesures dans le logiciel adéquat exécuté sur le PC. On pourra fournir le résultat sous la forme

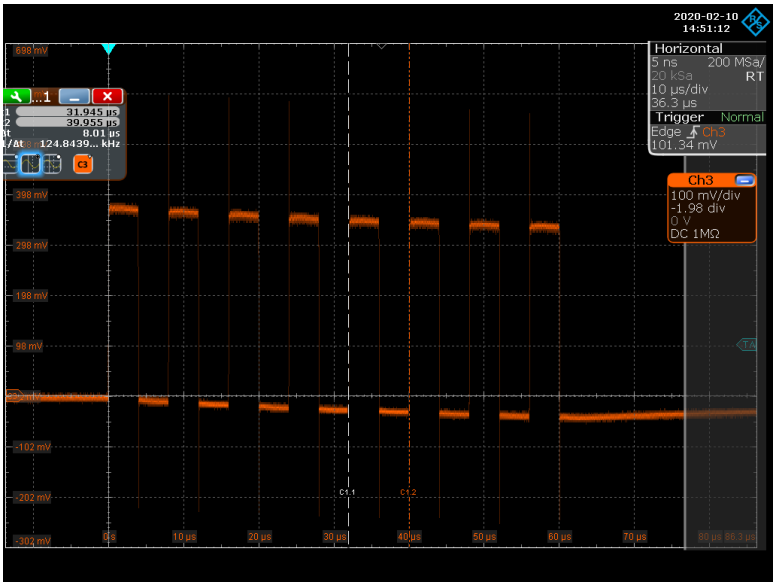
```
1A6E 1A5E 1A4E 1A3E 1A2E 1A1E 1A0E 19FE
2963 2973 2983 2993 29A3 29B3 29C3 29D3
...
```

7. Analyser la sortie que vous observez : est-elle cohérente avec vos attentes ? Expliquez/justifiez.
8. La résolution de la mesure est médiocre. Comment l'améliorer ? Démontrer.
9. Pourquoi cette méthode de mesure ne permet *pas* de détecter la dérive de fréquence du quartz cadençant le microcontrôleur ?
10. Comment modifier le montage pour mesurer la dérive en fréquence du quartz ?

1. Timer3 donc ICP3 est connecté à D13. SCK de SPI est sur D15 qui n'est accessible qu'au travers de la broche 9 de UEXT.
2. Le Makefile est au minimum de la forme

```
TARGET=main
CPU=atmega32u4
CFLAGS=-mmcu=$(CPU) -Os -Wall -I../../include -I../VirtualSerial/ -I../lufa-LUFA-140928 \
       -DF_USB=16000000UL -DF_CPU=16000000UL -Os -std=gnu99

all: $(TARGET).out

$(TARGET).out: $(TARGET).o spi.o input_capture.o
avr-gcc $(CFLAGS) -L../../../../../platforms/Atmega32/VirtualSerial -o $(TARGET).out \
                $(TARGET).o spi.o input_capture.o -lVirtualSerial
avr-objcopy -O ihex $(TARGET).out $(TARGET).hex

$(TARGET).o: $(TARGET).c
avr-gcc $(CFLAGS) -c $(TARGET).c

spi.o: spi.c
avr-gcc $(CFLAGS) -c spi.c

input_capture.o: input_capture.c
avr-gcc $(CFLAGS) -c input_capture.c
```

3. Exemple de capture d'écran :



La période est 8 μs en accord avec l'horloge (16 MHz) divisée par 128.

4. Un octet transmis sur SPI induit 8 transitions d'horloge sur SPI, donc nous remplaçons `res` par `res[8]`. Nous définissons un indice qui indique quelle case doit être remplie. La solution est donc

```
volatile short res[8];
volatile int indice;

ISR(TIMER3_CAPT_vect)
{if (indice<8)
    {res[indice]  = ICR3; indice++;}
}
```

Chaque mot du timer 3 est stocké sur 16 bits, en accord avec la taille de l'accumulateur du compteur.

5. Dans `main.c` :

```
extern int indice;
extern short res[8];

void transmit_data(uint8_t data)
{char buffer[2];
 buffer[0]=data;buffer[1]=0;
 fputs(buffer, &USBSerialStream);
}
```

```
void write_char(unsigned char c)
{if ((c>>4)>9) transmit_data((c>>4)-10+'A'); else transmit_data((c>>4)+'0');
 if ((c&0x0f)>9) transmit_data((c&0x0f)-10+'A'); else transmit_data((c&0x0f)+'0');
}

void write_short(unsigned short s)
{write_char(s>>8);
 write_char(s&0xff);
}

int main(void)
{ int k;
  indice=0;
 [...]
  USBCON=0;

 while(1)
  { _delay_ms(1000);
    sd_raw_send_byte(0x00); // emettre une trame SPI
    if (indice==8)
       {for (k=0;k<8;k++)
           {write_short(res[k]); transmit_data(0x20); }
        transmit_data(0x0A); transmit_data(0x0D);
        indice=0;
       }
    CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
    CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
    USB_USBTask();
  }
}
```

Nous avons donc exporté `indice` en préfixant sa déclaration de `extern`.

6. Le résultat est, sous `minicom` :

```
1A6E 1A5E 1A4E 1A3E 1A2E 1A1E 1A0E 19FE
2963 2973 2983 2993 29A3 29B3 29C3 29D3
6D3A 6D4A 6D5A 6D6A 6D7A 6D8A 6D9A 6DAA
48B5 48A5 4895 4885 4875 4865 4855 4845
04E1 04D1 04C1 04B1 04A1 0491 0481 0471
3EF6 3F06 3F16 3F26 3F36 3F46 3F56 3F66
772C 771C 770C 76FC 76EC 76DC 76CC 76BC
```

7. Les valeurs s'incrémentent le long de chaque ligne de 0x10, soit 16. Il s'agit bien du ratio entre le pre-scale cadençant SPI (/128) et le pre-scaler cadençant le timer (/8).
8. Amélioration de la résolution en réduisant le pre-scaler de 8 à 1 sur le timer. Dans ce cas

```
55BE 553E 54BE 543E 53BE 533E 52BE 523E
2AC6 2A46 29C6 2946 28C6 2846 27C6 2746
000C 0074 00F4 0174 01F4 0274 02F4 0374
2AAC 2B2C 2BAC 2C2C 2CAC 2D2C 2DAC 2E2C
```

   et cette fois le timer s'incrémente de 128 entre deux mesures, soit 0x80. En effet (première ligne) 0x30+0x80=0xB0 (puisque 3+8=11=0xb) qui est bien la valeur observée.
9. Le même quartz cadence les deux périphériques dont la contribution de sa dérive est éliminée dans cette mesure relative.
10. Déclencher *input capture* par une source externe stables de temps, e.g. 1-PPS de GPS.