Sommaire

Historique

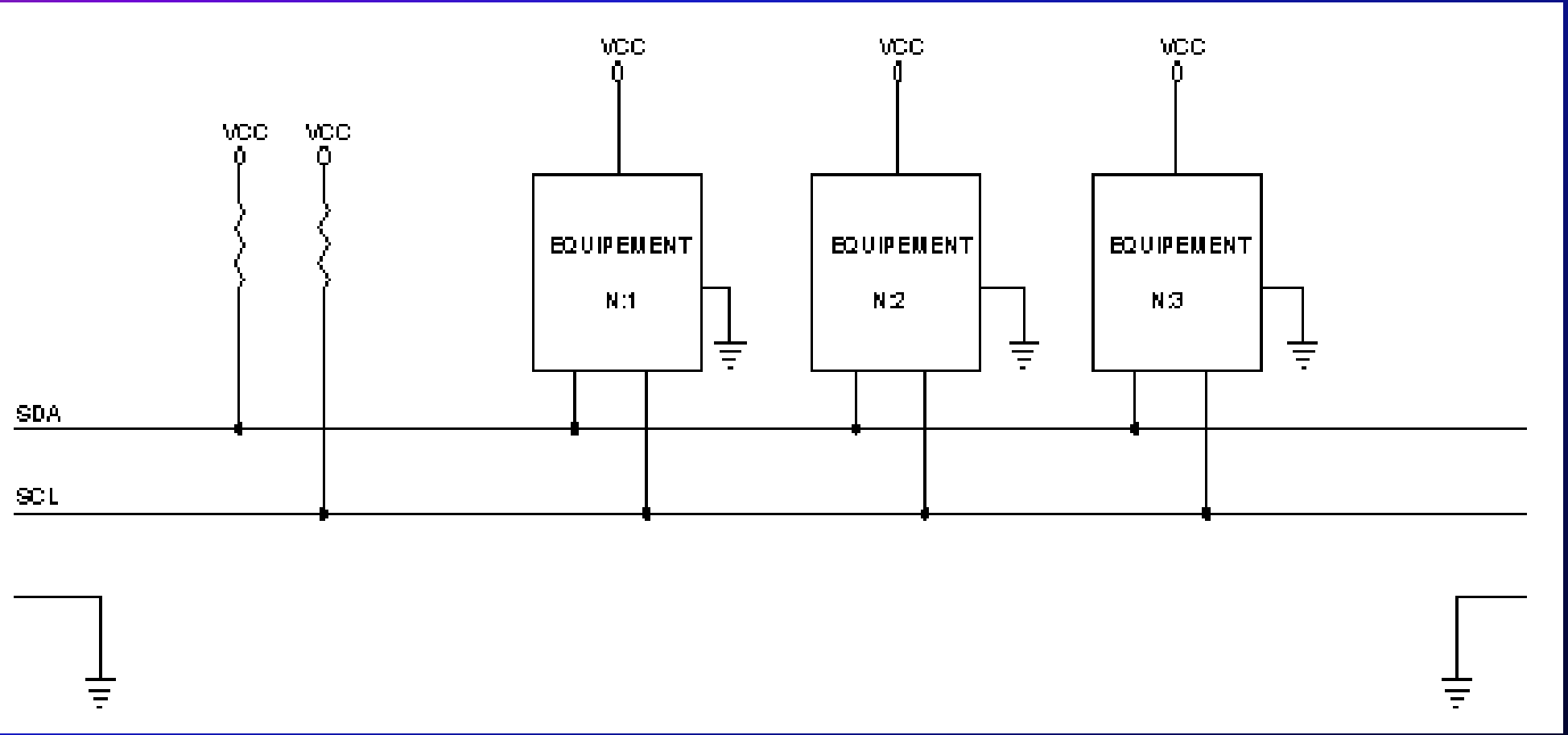
- But : faire communiquer à haute vitesse des composants électroniques de fonctions diverses (potentiomètres, RTC, mémoires, etc.) et d'origines diverses (Philips, Analog Device, Maxim, etc.) à l'aide d'un protocole standardisé simple et en limitant le câblage.
- Réalisation : au début des années 80 Philips met au point les composants qui communiquent par une ligne de données SDA et une ligne d'horloge SCL. En 1982 la norme "**I**nter **I**ntegrated **C**ircuit" est déposée. Aujourd'hui la division semi-conducteurs de Philips est NXP qui a publié la version 4.0 en 2012.

Terminologie

Terminologie

- **SDA** : Serial Data = ligne de transmission des données
- **SCL** : Serial Clock = ligne du signal d'horloge
- **Émetteur** : composant qui envoie des données sur le bus
- **Récepteur** : composant qui reçoit les données présentes sur le bus
- **Maître** : composant qui prend le contrôle du bus. Il génère l'horloge. Le maître peut être émetteur ou récepteur.
- **Esclave** : composant adressé par le maître. L'esclave peut être émetteur ou récepteur.
- **Multi-maître** : plusieurs maîtres peuvent tenter de contrôler le bus
- **Arbitrage** : procédure permettant de choisir un seul maître alors que plusieurs tentent de prendre le contrôle du bus
- **Synchronisation** : synchronisation des horloges de plusieurs maîtres

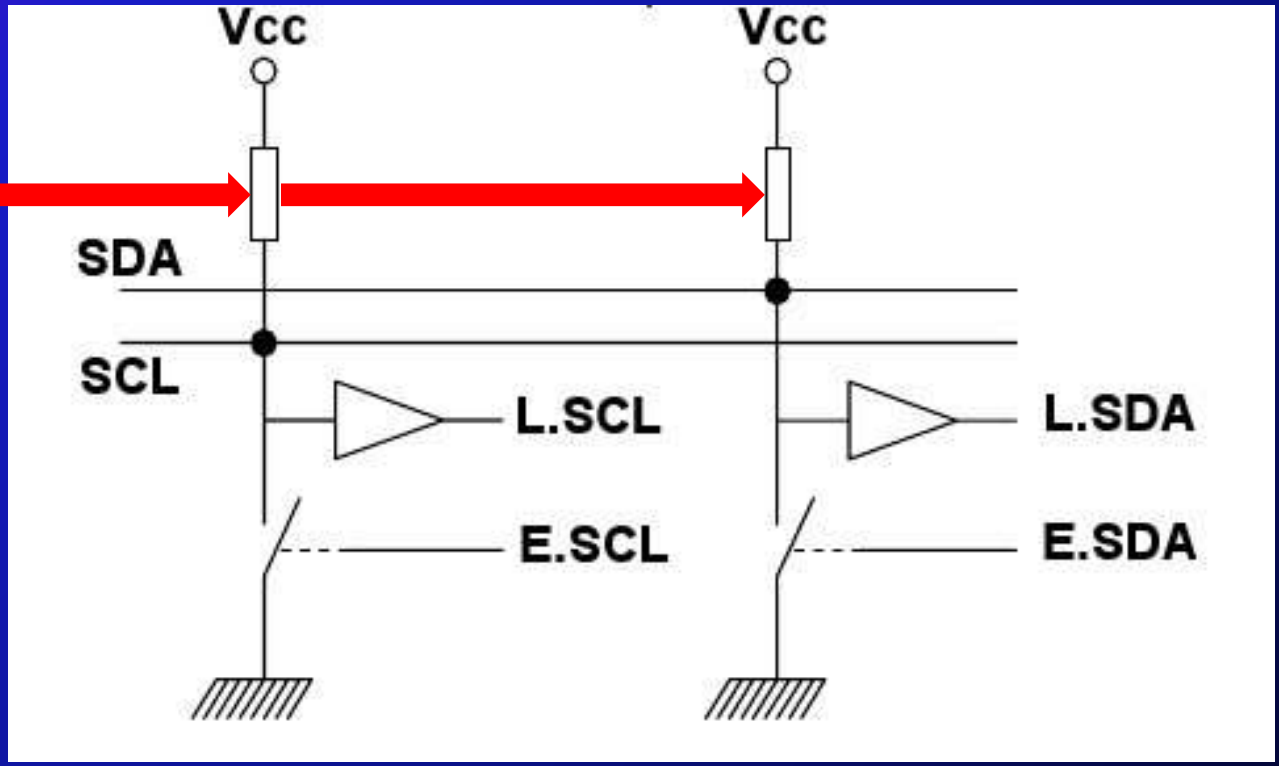
Périphériques i²C en bus



Les données sont envoyées en série sur la ligne SDA. La ligne SCL est l'horloge
Chaque équipement dispose d'une adresse unique

E/S de type collecteur ou drain ouvert

ATTENTION :
prévoir des résistances de pull-up (tirage) entre Vcc et SDA et Vcc et SDL.
Généralement 4.7 kΩ



Avantages :

- Architecture ultra simple
- Les lignes SDA et SCL sont au repos à l'état haut. Elles peuvent être forcées uniquement à l'état bas.
- Pas de conflit électrique

Adressage

Adressage

- Les adresses sont codées sur 7 bits
- Les adresses sont attribuées par le "i²C Committee" en fonction du type de périphérique
- Le 8ième bit indique que l'opération suivante sera une écriture (0) ou une lecture (1)
- Lors d'une lecture le maître connait le nombre d'octets qu'il attend

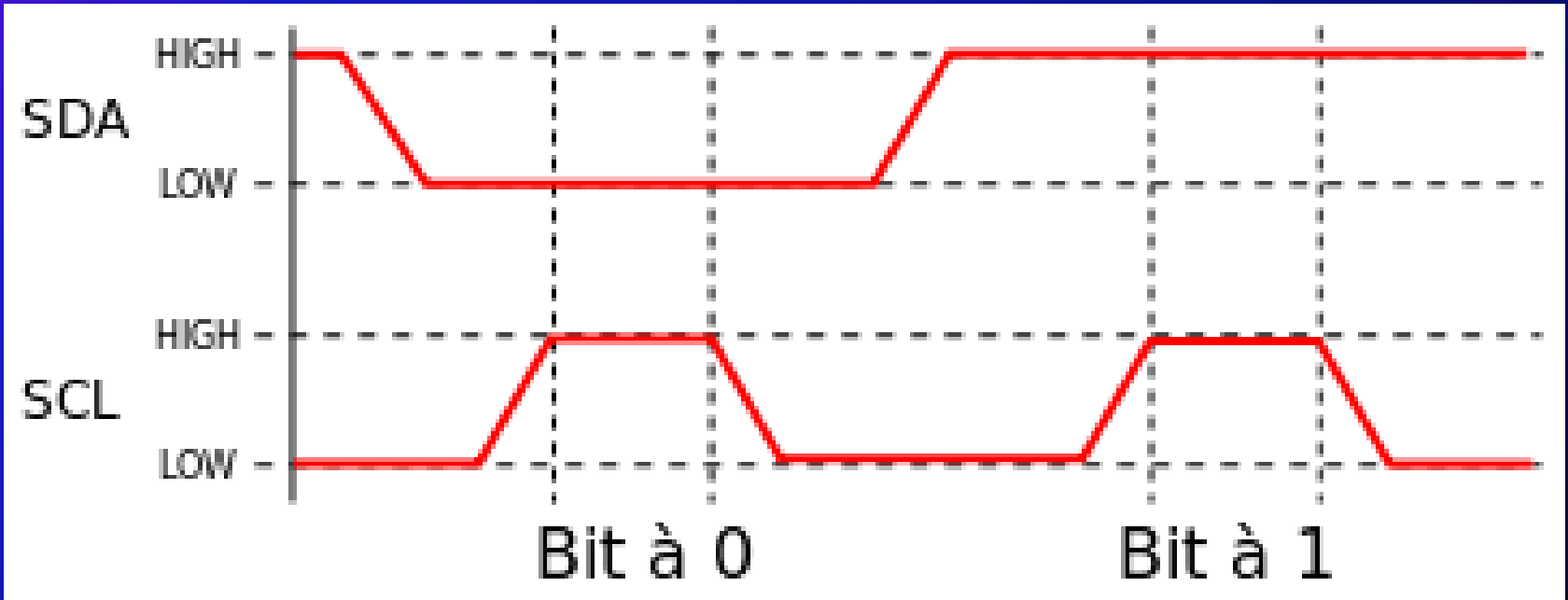
Adresses réservées : les adresses 0000 0xxx ne sont jamais attribuées à des composants

- 0000 0000 : appel général. Tous les périphériques renvoient un acquittement
- 0000 0110 : reset. Tous les périphériques retournent à leur état initial
- 0000 0010 : tous les périphériques reprennent leur adresse initiale
- 0000 0100 : comme ci-dessus mais pour les circuit dont on peut définir l'adresse matériellement

Codage

Codage de bits : **NRZ** MSB first. Le niveau (« HIGH » ou « LOW ») de la ligne SDA doit être maintenu stable pendant le niveau « HIGH » sur la ligne SCL pour la lecture du bit.

Cette règle pourra être outrepassée pour des bits de contrôle.

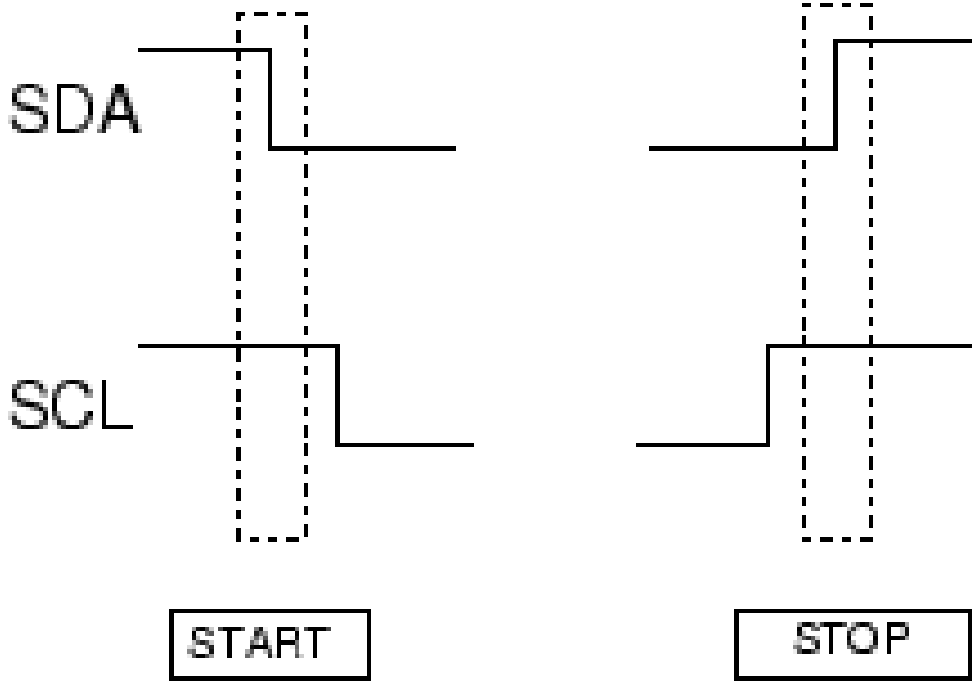


Accès au bus

Accès au bus

- Les lignes SDA et SCL sont à 1 lorsque le bus est disponible (idle)
- Le bus doit être "idle" depuis plus de 4.7µs pour prétendre en prendre le contrôle
- Pour prendre le contrôle du bus SDA passe à 0 et SCL reste à 1 (START)
- Le périphérique qui prend le contrôle du bus en devient le maître
- C'est le maître qui génère l'horloge (SCL)
- Le bus est libéré lorsque SDA passe à 1 et SCL reste à 1 (STOP)
- Les conditions de START et de STOP ne sont générées que par le maître

Conditions Start / Stop

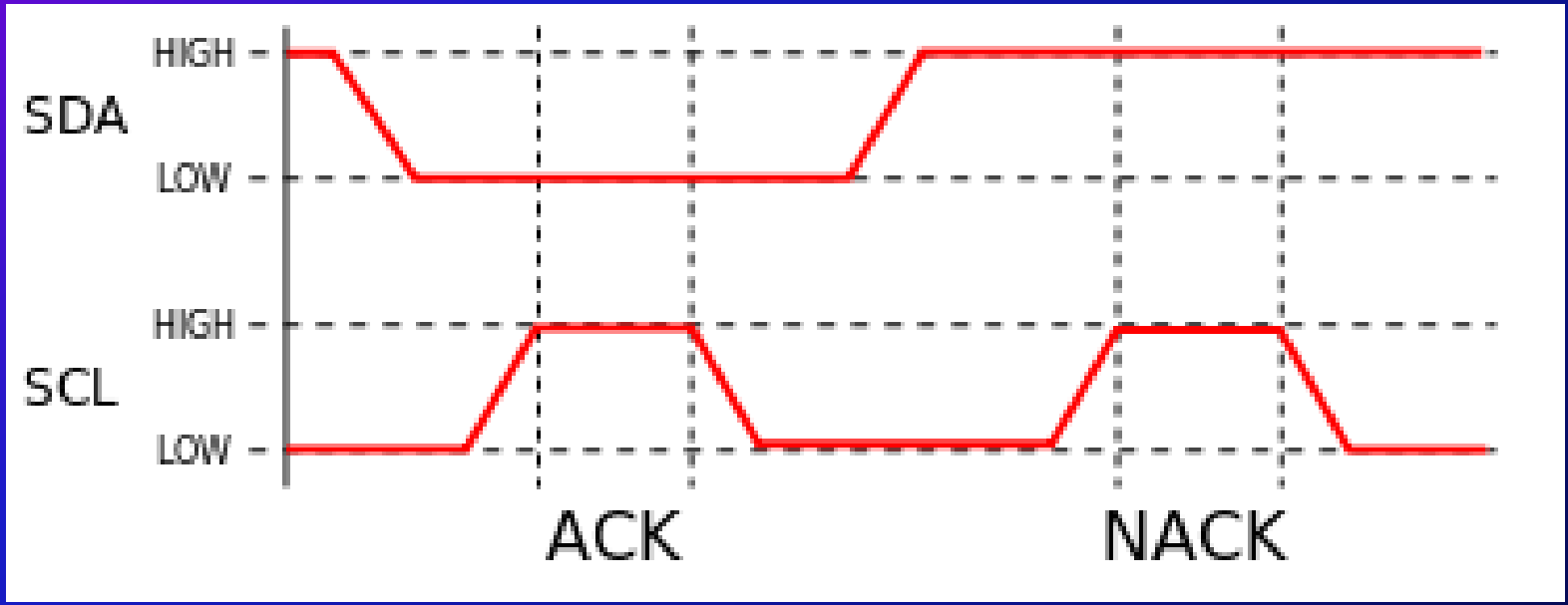


Connexion maître / esclave

Connexion du maître à l'esclave

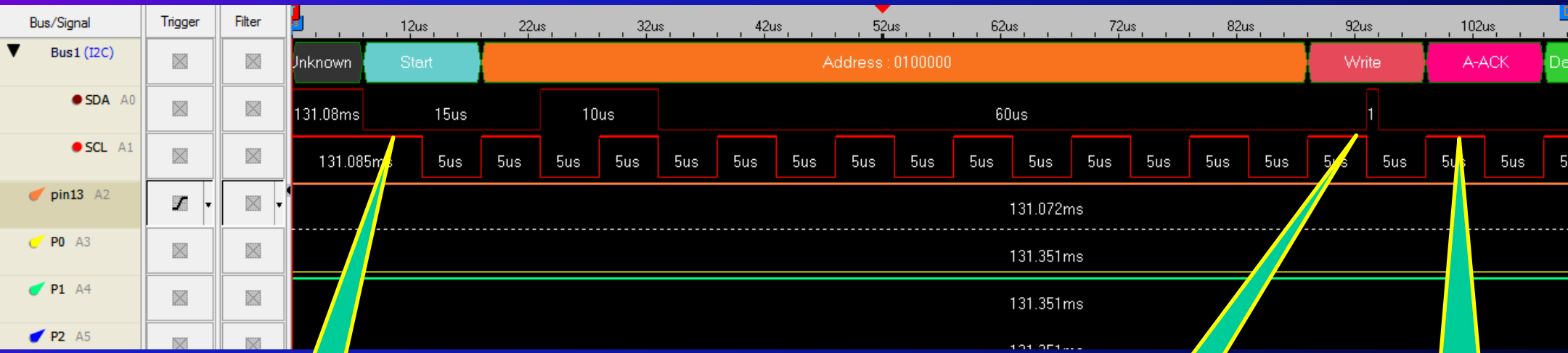
- Le maître envoie les 7 bits d'adresses + le bit R/W soit 1 octets
- Après avoir reçu l'adresse correctement le récepteur accuse réception (**A**ddress-**ACK**nowledgment) en passant SDA à 0 pendant que SCL est à 1
- Le maître envoie ensuite un octet de données
- La bonne réception d'un octet de données est de même acquitté par un **D**ata-**ACK**nowledgment.
- Si nécessaire un autre octet de données est envoyé puis acquitté
- Clôture de la connexion par une condition STOP

Acquittement



En cas d'erreur d'acquittement c'est une condition **No-ACK**nowledgment qui est envoyée

Prise de contrôle du bus en écriture

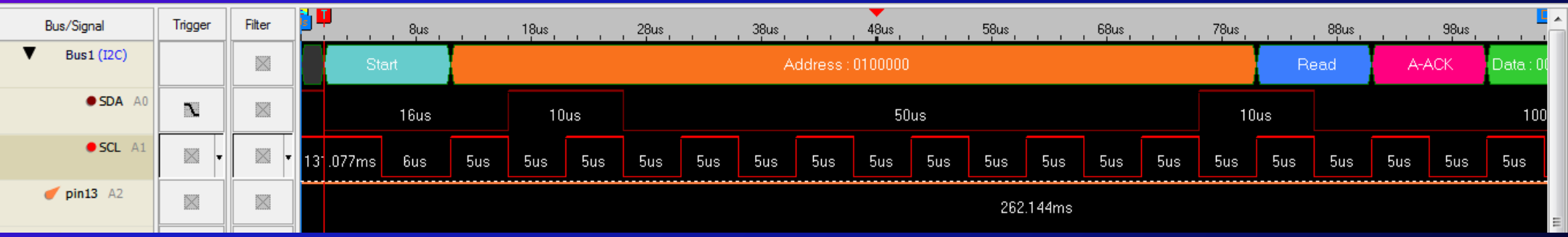


Condition de start

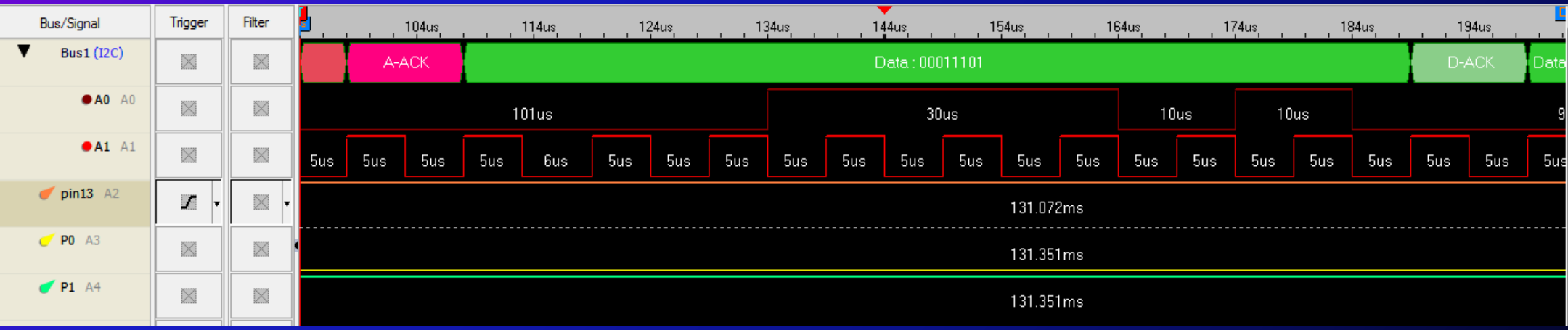
Bit d'écriture

Acquittement

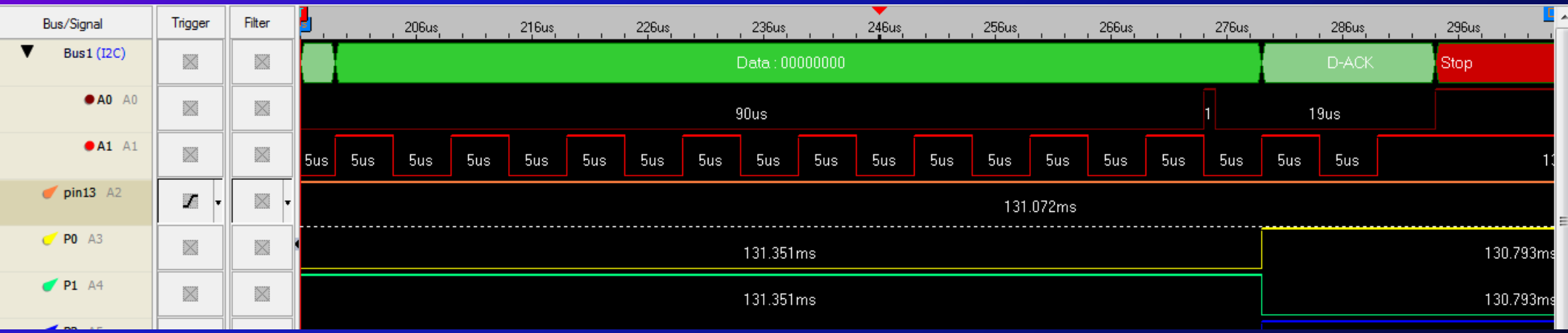
Prise de contrôle du bus en lecture



Transmission de données



Libération du bus



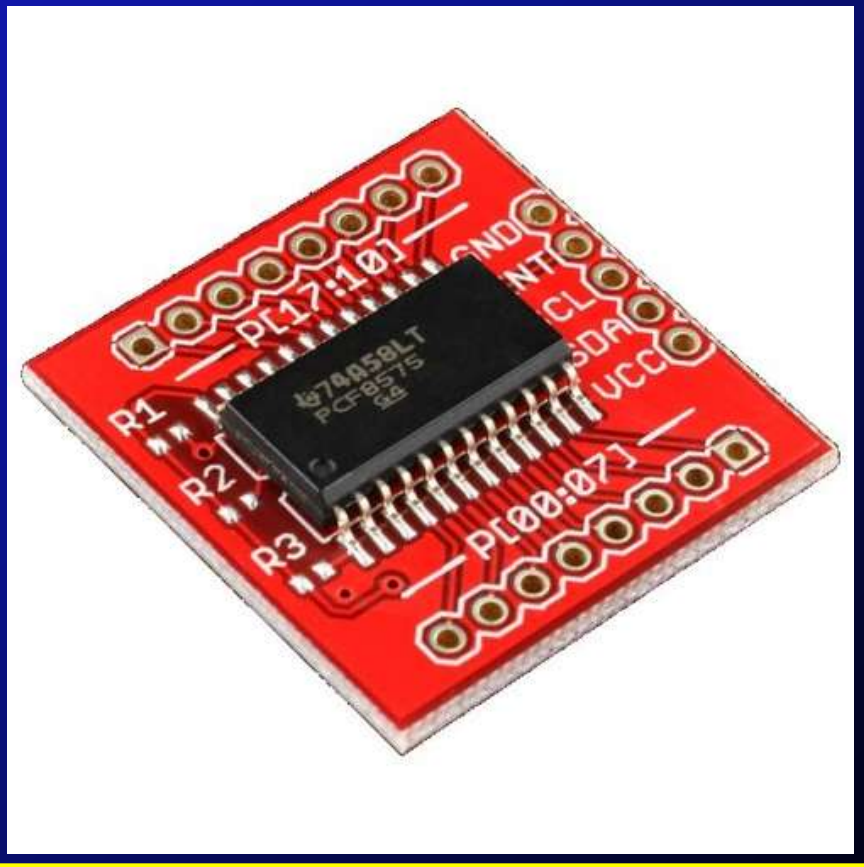
Le composant PCF8575

Ce composant de Texas Instruments permet de lire ou d'écrire à partir d'un bus I2C 16 E/S digitales.

Nous utilisons une platine d'expérimentation de Sparkfun (disponible chez Lextronic).

Remarques :

- Les résistances de tirage sont intégrées
- Par défaut l'adresse est 0x20. Elle peut être changée en faisant un pont de soudure sur les contacts A0, A1,A2 sous la plaque



Montage écriture

- Connecter la carte sur l'Arduino
- Connecter les leds aux broches 0, 2, 3 et 4.Déterminer la valeur des résistances à utiliser

Programme Arduino écriture (setup)

Ce programme fait clignoter au rythme de 0.2s les LED branchées sur les sorties 0, 2, 3 et 4. La LED 1 clignote au même rythme mais ses phases d'allumage / extinction sont inversées.

```
#define PCF8575_ADDR 0x20     // Adresse par défaut
#define TRIGG_LED 13          // Trigger pin
#include <Wire.h>                     // Chargement de la librairie I2C

void setup()
{
   pinMode(TRIGG_LED, OUTPUT);        // Sera utilisé comme trigger
   Wire.begin();                      // Démarrage de l'interface I2C
}
```

Programme Arduino écriture (loop)

```
void loop()
{
  digitalWrite(TRIGG_LED, HIGH);                  // Armement du trigger
  delayMicroseconds(5);                           // Temporisation
  Wire.beginTransmission(PCF8575_ADDR);           // Début de transmission
  Wire.write(0b00011101);                         // pin 0 2 3 4 HIGH
  Wire.write(0);                                  // pin 8->15 LOW
  Wire.endTransmission();                         // Stop transmission
  delay(200);                                     // Demi période de clignotement
  digitalWrite(TRIGG_LED, LOW);                   // Désarmement du trigger
  delayMicroseconds(5);                           // Temporisation
  Wire.beginTransmission(PCF8575_ADDR);           // Début de transmission
  Wire.write(0b00000010);                         // pin 1 HIGH
  Wire.write(0);                                  // pin 8->15 LOW
  Wire.endTransmission();                         // Stop transmission
  delay(200);                                     // Demi période de clignotement
}
```

Analyse écriture

- Montrer le trafic sur le bus i2c

Montage lecture

- Connecter la pin Int du PCF8575 à la pin 2 de l'Arduino
- Connecter un bouton anti rebond à la pin 8 du PCF8575. Le bouton est à l'état haut au repos.

Programme Arduino lecture (setup)

```
#define PCF8575_ADDR      0x20
#define PCF8575_INT_PIN   2
#define TRIGG_LED         13
#include <Wire.h>

void setup()
{
  pinMode(TRIGG_LED, OUTPUT);
  pinMode(PCF8575_INT_PIN, INPUT);
```

Programme Arduino lecture (setup suite)

Mise à 0 de toutes les pin du PCF8575

```
  Wire.begin();                                  // Initialisation
  delay(100);                                          // Attendre un peu
  Wire.beginTransmission(PCF8575_ADDR);          // Début de transmission
  Wire.write(0);                                 // pin 0-7 LOW
  Wire.write(255);                               // pin 8-15 HIGH
  Wire.endTransmission();                        // Fin de transmission
  delay(100);
   // La pin Int du PCF8575 change d'état lorsque une des pin 0-15 change d'état
   // Lorsque la pin Int passe de HIGH à LOW on appelle la fonction intPCF32
  attachInterrupt(PCF8575_INT_PIN, intPCF32, FALLING);
}
```

Programme Arduino lecture (interruption)

```
void loop()
{ // Ne fait rien : attente de l'interruption }

void intPCF32()
{
  Wire.requestFrom(PCF8575_ADDR, 2); // Attente de 2 octets
  byte tampon[2];
  int i=0;
  while(Wire.available())            // Lecture des 2 octets
  { tampon[i] = Wire.read();
    Serial.println(tampon[i]);
    i++; }
}
```

Analyse lecture

- Montrer les échanges sur le bus

Composant MAXIM DS 3231

Le composant MAXIM DS 3231

- Ce composant est une horloge temps réel (RTC) intégrant un quartz à 32.768KHz et deux alarmes programmables.
- L'accès se fait par la lecture/écriture d'octets identifiés par des registres. Ex : les registres 0x00 à 0x06 contiennent respectivement les secondes, minutes, heure, jour, date, mois et année (soit les 7 registres). Il n'y a pas de contrôle de cohérence du jour et de la date
- L'année est sur deux chiffre avec le positionnement du bit 7 du registre 0x05 à 1 au changement de siècle. L'horloge tient compte des années bissextiles mais boguera le 27/02/2100 (ce jour n'existe pas).
- Le codage est en BCD
- L'adresse est $104_{10}=68_{16}=110\ 1000_2$
- Le composant est toujours contacté d'abord en mode écriture en indiquant sur quel registre on veut agir. Les valeurs des registres 0x00 à 0x06 sont dupliquées pour qu'il n'y ait pas d'altération lorsque l'horloge évolue. On peut ensuite lire ou écrire en séquence à partir du registre choisi.

Montage d'expérimentation

Descriptif montage

Pour le montage d'expérimentation nous avons utilisé les éléments suivants :

- Une carte Arduino Uno maître i²C
- Un shield "Le Sablier" Snootlab. équipé de la RTC MAXIM DS3231 esclave i²C
- Un analyseur de protocole ZeroPlus LAP-C(16128)
- Logiciel d'analyse Logic Cube

"Le Sablier" est connecté au bus i²C de la carte Arduino. L'appui sur le bouton fait basculer le bit INTCN (bit 2 du registre 0x0e) ce qui active ou désactive la génération d'un signal carré à la fréquence ¼ de l'oscillateur sur la broche SQW. La broche 13 de l'Arduino sert au déclenchement de l'analyse. Sur un front descendant : write et read sur un front montant. On a donc les broches suivantes analysées :

- Broches SDA et SCL
- L'oscillateur à 32.768 KHz
- Broche SQW (Square Wave)
- Trigger broche 13

Le bon fonctionnement se vérifie à l'allure du signal sur la broche SQW

Montage de la platine de prototypage

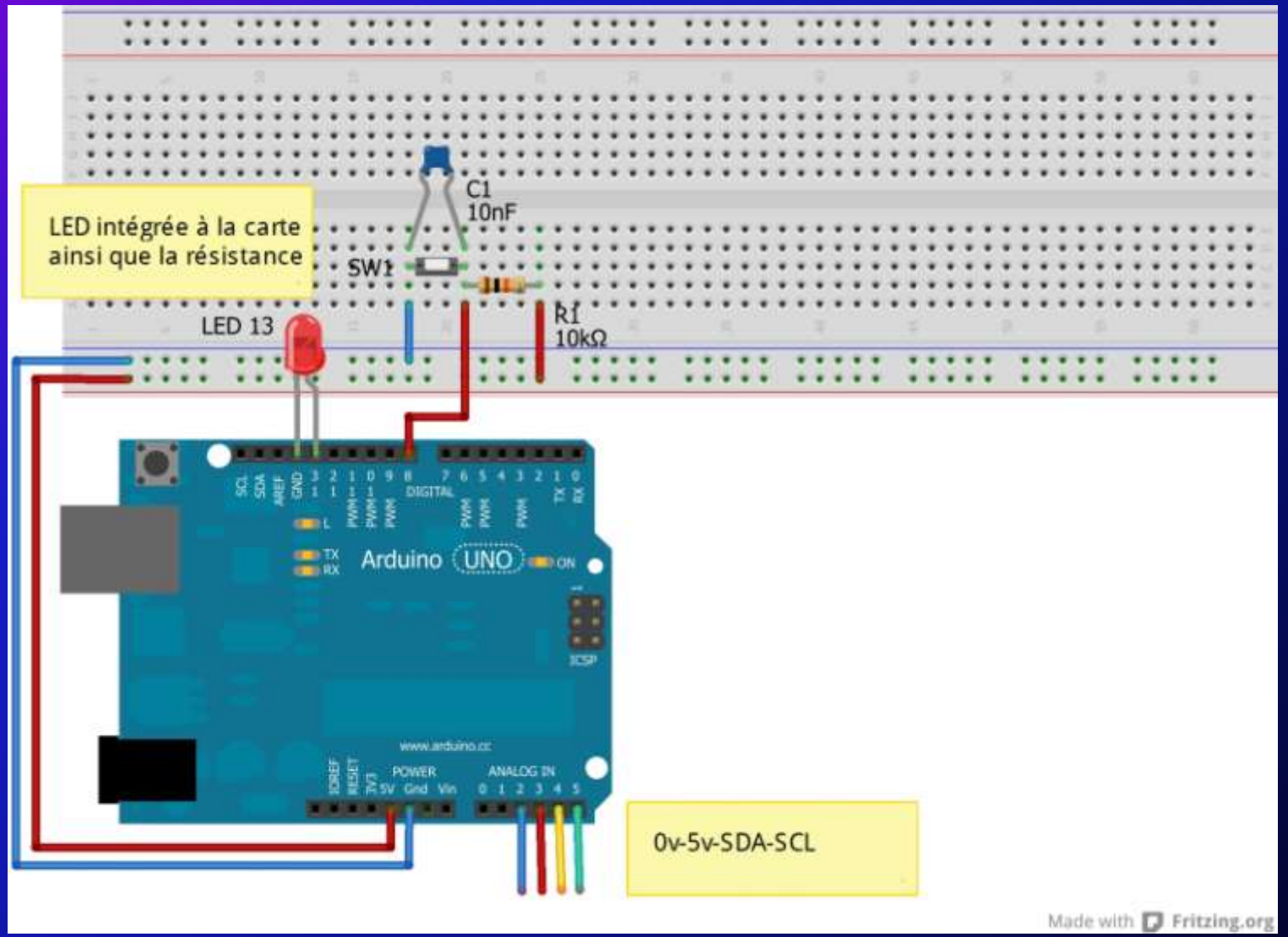
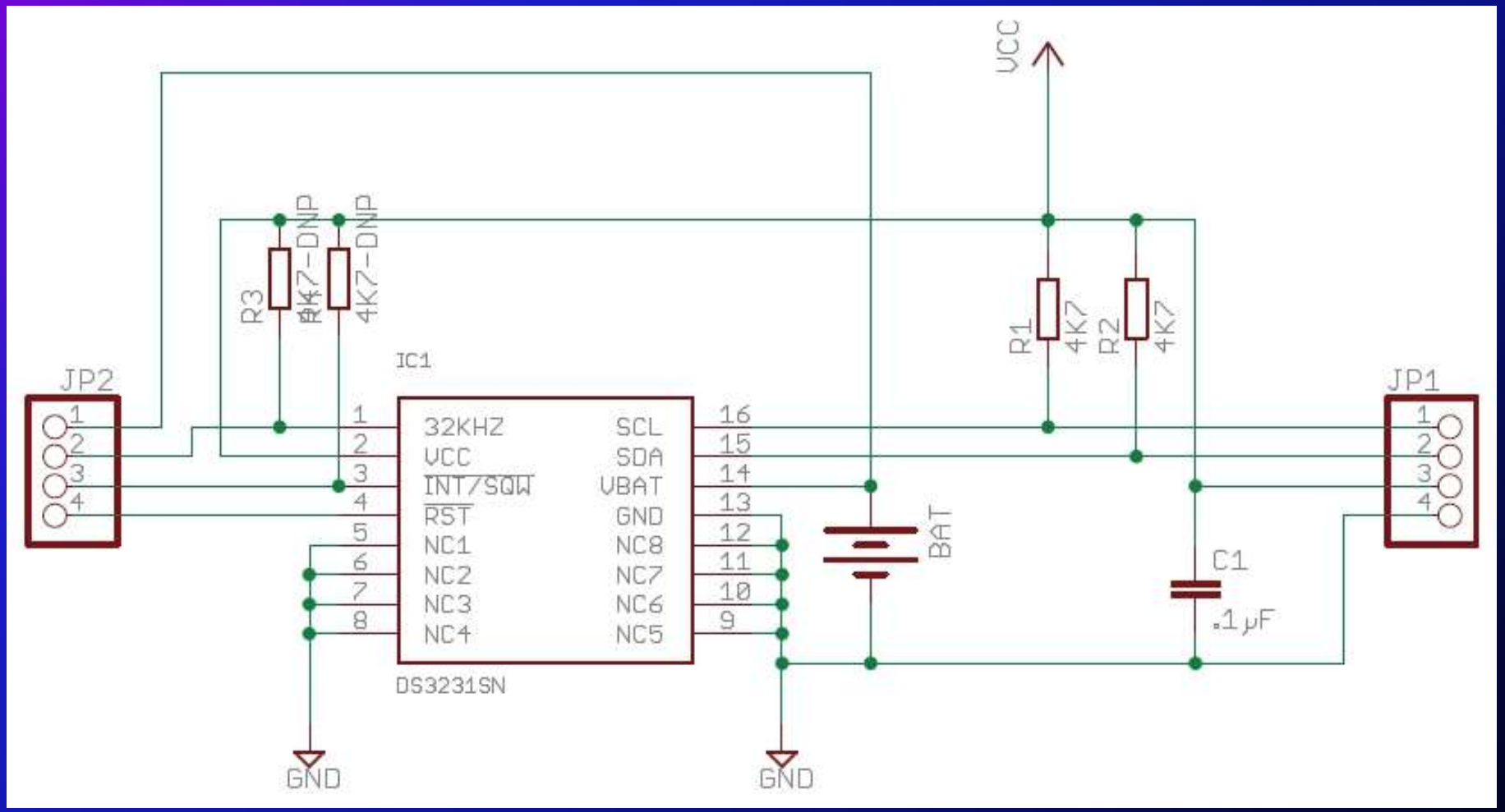
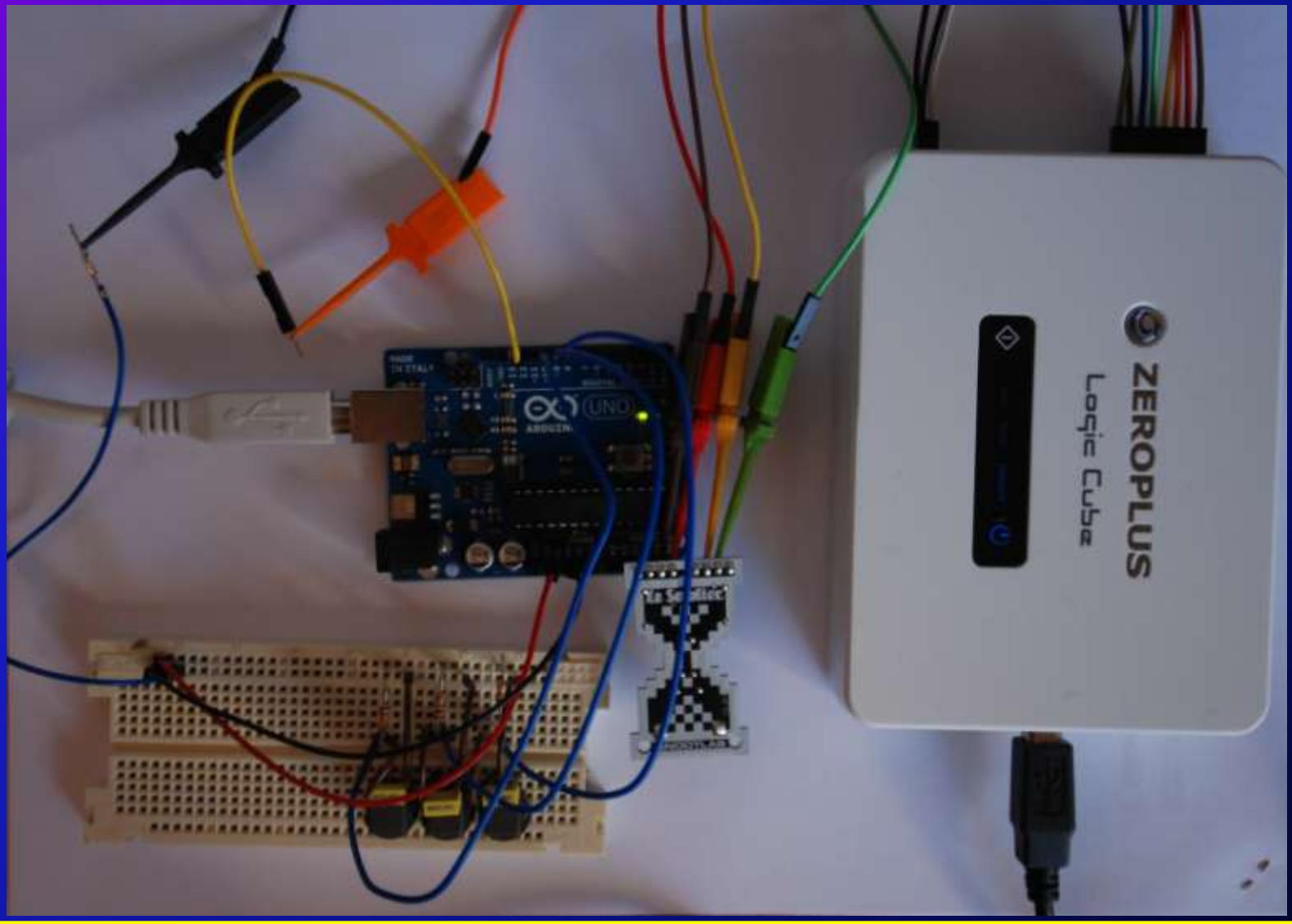


Schéma de câblage du shield "Le Sablier" Snootlab

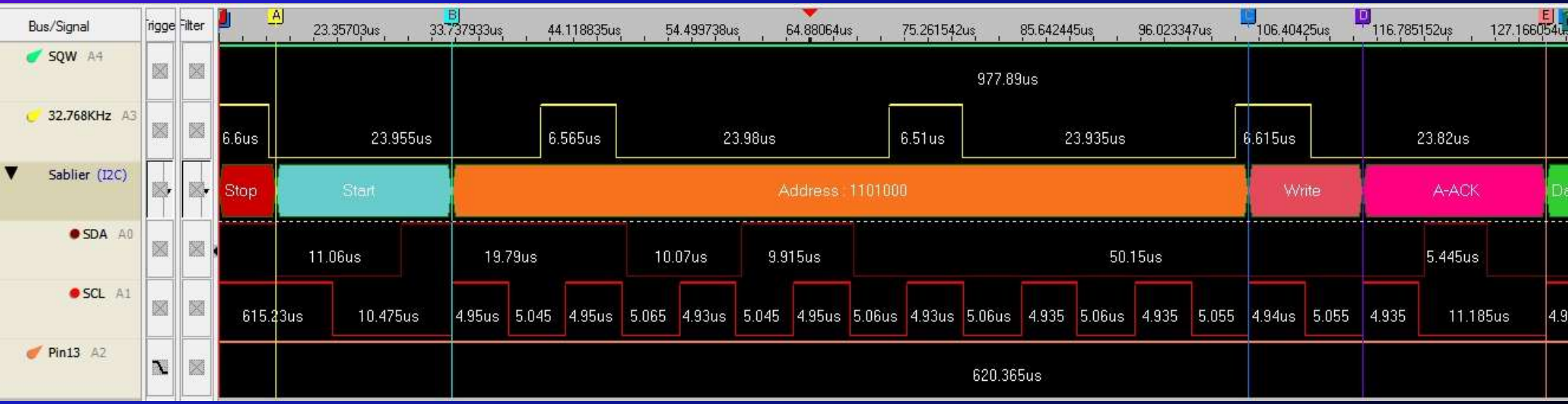


La réalité …



Prise de contrôle du bus en écriture

- STOP : le bus est libre SDA=1 et SCL=1
- Prise de contrôle du bus SDA=0
- Condition de START
- Envoie des 7 bits d'adresse 1101 000
- Envoie du bit W=0 soit l'octet envoyé : 1101 0000
- Acquittement envoyé de l'esclave au maître



Prise de contrôle du bus en lecture

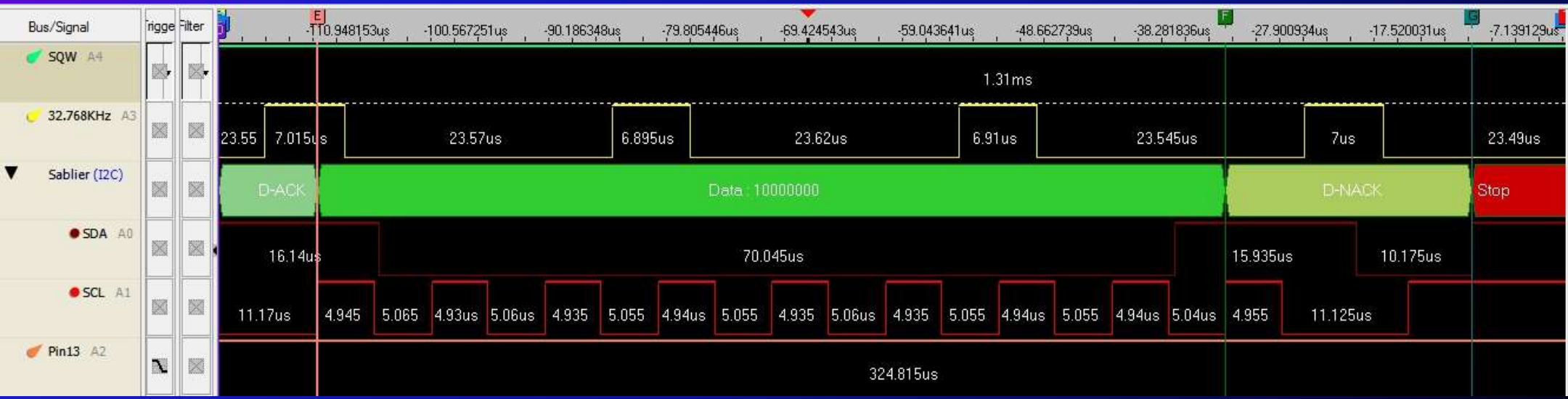
Le processus est le même sauf le dernier bit READ=1 soit l'octet envoyé :1101 0001

Écriture de données

- À la fin de chaque octet écrit l'esclave envoie au maître un D-ACK
- Quand le maître n'a plus de données à transmettre il libère le bus : condition STOP

Lecture de données

- À la fin de chaque octet lu le maître envoie à l'esclave un D-ACK
- Après le dernier octet reçu (le maître connait le nombre d'octets attendus) il envoie un D-NACK (NACK=no ACK) à l'esclave puis il libère le bus : condition STOP



TP 1

- À partir du fichier sablier5w1.alc déterminer :
  - La fréquence moyenne de la ligne SCL en MHz calculée sur les 8 périodes de l'envoie du premier octet de données
  - Le débit moyen en M bits / s de la ligne SDA pour l'envoie du second octet de données
  - Entre les deux états STOP exclus du bus i²C, reporter sur deux graphes (un pour l'état haut et un pour l'état bas) la variation temporelle de l'oscillateur avec en abscisse le n° d'ordre de l'état et en ordonnée l'écart par rapport à la moyenne (que vous aurez préalablement calculée) de la durée de l'état à l'échelle 5cm / 1 µs. Vous ferez apparaitre les dispersions maximales pour les deux états
  - Quelle est la fréquence moyenne mesurée de l'oscillateur ? Conclusion ?
  - Montrer avec un octet de votre choix que vous préciserez que le codage est bien du type NRZ
  - Lors de l'écriture de l'adresse, le premier changement d'état de SDA intervient combien de temps après que SCL soit passée à 0
  - Quelle est l'adresse du périphérique en décimal
  - Quel est le registre qui va être écrit en hexadécimal
  - Quelle est la valeur du bit 2 (LSB=bit 0 à) qui va être écrite

TP 2

- À partir du fichier sablier5w0.alc
  - Quelle est la donnée qui a été changée
  - Quelle influence sur SQW
  - À quel moment le changement de comportement de SQW intervient
  - Quelle est la fréquence de SQW que vous pouvez déduire visuellement de la fréquence de l'oscillateur
- À partir du fichier sablier5.ino
  - Installer ce programme sur la carte Arduino et faire les connexions avec l'analyseur logique ZeroPlus
  - Tester en changeant le trigger associé à la pin 13. Valider le bon fonctionnement
  - Comment diviser par deux la fréquence de SQW (datasheet p 11 et p 13)
  - Modifier le programme Arduino pour :
    - Maintenir un signal carré sur SQW
    - L'appui sur le bouton 8 passe alternativement la fréquence de SQW de 8.192 KHz à 4.096 KHz et réciproquement
  - Justifier votre résultat

Fonctionnement multi-maîtres