Session Hiver 2019

Pascal-Emmanuel Lachance

Cégep Limoilou

[Adresse de la société]

CARNET DE BORD   
PASCAL-EMMANUEL LACHANCE

Table des matières

[CARNET DE BORD PASCAL-EMMANUEL LACHANCE 0](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8718483)

[Table des illustrations 4](#_Toc8718484)

[Semaine 9 5](#_Toc8718485)

[Objectifs 5](#_Toc8718486)

[Résultats 5](#_Toc8718487)

[Lundi, 13 Mai 2019 5](#_Toc8718488)

[Nous n’avions pas compris le principe d’un diviseur de tension 5](#_Toc8718489)

[Pas besoin de changer le boost converter! 5](#_Toc8718490)

[Changement du boost converter 6](#_Toc8718491)

[Semaine 8 7](#_Toc8718492)

[Objectifs 7](#_Toc8718493)

[Résultats 7](#_Toc8718494)

[Vendredi, 10 Mai 2019 7](#_Toc8718495)

[Resoudure FPGA 7](#_Toc8718496)

[Resoudure connecteur écran 7](#_Toc8718497)

[Problèmes de timing avec signal SHIFT 7](#_Toc8718498)

[Short sur l’alimentation de l’écran 7](#_Toc8718499)

[Jeudi, 9 Mai 2019 8](#_Toc8718500)

[Test tensions diviseur 8](#_Toc8718501)

[2 patchs 8](#_Toc8718502)

[1 patch? 8](#_Toc8718503)

[Condensateurs 0603 pour version 1.1 8](#_Toc8718504)

[Tests de l’écran avec chat 8](#_Toc8718505)

[Alimentation shutdown 8](#_Toc8718506)

[Data à 200mV 8](#_Toc8718507)

[Mercredi, 8 Mai 2019 8](#_Toc8718508)

[Correction short 8](#_Toc8718509)

[Premier test! 8](#_Toc8718510)

[Inspections niveaux de tension 8](#_Toc8718511)

[Xilinx ISE sur Windows 7 32-bits 8](#_Toc8718512)

[Programmation de notre carte par JTAG! 8](#_Toc8718513)

[Module SPI C & VHDL 8](#_Toc8718514)

[Mardi, 7 Mai 2019 8](#_Toc8718515)

[Inspection 8](#_Toc8718516)

[Short sur 1.2V 8](#_Toc8718517)

[Fin de la soudure, bottom layer 8](#_Toc8718518)

[Lundi, 6 Mai 2019 9](#_Toc8718519)

[Soudure 9](#_Toc8718520)

[Mise dans le four 9](#_Toc8718521)

[Mauvais padstacks condensateurs 4.7µF 9](#_Toc8718522)

[Footprint switchés 9](#_Toc8718523)

[Semaine 7 10](#_Toc8718524)

[Objectifs 10](#_Toc8718525)

[Résultats 10](#_Toc8718526)

[Dimanche, 5 Mai 2019 10](#_Toc8718527)

[Plus d’ajustements sur le boîtier 10](#_Toc8718528)

[Test du port parallèle VHDL 10](#_Toc8718529)

[Samedi, 4 Mai 2019 11](#_Toc8718530)

[Ajustements sur boîtier 11](#_Toc8718531)

[Vendredi, 3 Mai 2019 11](#_Toc8718532)

[Tests UART 11](#_Toc8718533)

[Réception de nos PCBs! 12](#_Toc8718534)

[Nouveau logo 13](#_Toc8718535)

[Jeudi, 2 Mai 2019 13](#_Toc8718536)

[Switch codeurs 13](#_Toc8718537)

[Tests I2C 13](#_Toc8718538)

[LCD\_Write\_I2C 13](#_Toc8718539)

[Mercredi, 1er Mai 2019 16](#_Toc8718540)

[UART pour version 1.1 16](#_Toc8718541)

[Mardi, 30 avril 2019 17](#_Toc8718542)

[Modèle 3D du boîtier 17](#_Toc8718543)

[Nos PCBs arrivent jeudi! 17](#_Toc8718544)

[Chat envoyé par port parallèle 18](#_Toc8718545)

[Semaine 6 19](#_Toc8718546)

[Objectifs 19](#_Toc8718547)

[Résultats 19](#_Toc8718548)

[Dimanche, 28 avril 2019 19](#_Toc8718549)

[Entête de fonction 19](#_Toc8718550)

[Entête de LCD\_Init\_I2C 19](#_Toc8718551)

[Samedi, 27 avril 2019 19](#_Toc8718552)

[Ajustements port parallèle 19](#_Toc8718553)

[Vendredi, 26 avril 2019 20](#_Toc8718554)

[Port parallèle dans la librairie LCD 20](#_Toc8718555)

[LCD\_Write\_Parallel 20](#_Toc8718556)

[Jeudi, 25 avril 2019 22](#_Toc8718557)

[Test du port parallèle 22](#_Toc8718558)

[Parallel.h 23](#_Toc8718559)

[Parallel.c 24](#_Toc8718560)

[Reflow Profile de notre solder paste 31](#_Toc8718561)

[Réception de nos pièces! 31](#_Toc8718562)

[Mercredi, 24 avril 2019 32](#_Toc8718563)

[Variadic functions 32](#_Toc8718564)

[Port parallèle sur STM 32](#_Toc8718565)

[Data : 32](#_Toc8718566)

[Adress 32](#_Toc8718567)

[ChipSelect 32](#_Toc8718568)

[Mardi, 23 avril 2019 32](#_Toc8718569)

[Commande des pièces 32](#_Toc8718570)

[Allocation de mémoire dynamique 32](#_Toc8718571)

[Port parallèle STM32 33](#_Toc8718572)

[Lundi, 22 avril 2019 33](#_Toc8718573)

[Librairie STM32 33](#_Toc8718574)

[Plugin VisualGDB 34](#_Toc8718575)

# Table des illustrations

[Figure 1 - Courant de sortie du régulateur AP3012 5](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667726)

[Figure 3 - Typical application circuit MIC2288 5](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667727)

[Figure 2 - Courant de sortie du MIC2288 5](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667728)

[Figure 4 - Ajustements sur trous de vis 9](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667729)

[Figure 5 - Boîtier avec composantes through-hole installés sur le PCB. 10](#_Toc8667730)

[Figure 6 - PCB reçu, top face 11](#_Toc8667731)

[Figure 7 - PCB reçu, bottom face 11](#_Toc8667732)

[Figure 8 - Nouveau logo GraphBit 12](#_Toc8667733)

[Figure 9 - Rajout d'un connecteur pour le UART 15](#_Toc8667734)

[Figure 10 - Modèle 3D ajusté 16](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667735)

[Figure 11 - Shipment tracking 16](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667736)

[Figure 12 - Premier test du port parallèle 19](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667737)

[Figure 13 - Exemple de configuration du port parallèle 21](file:///C:\Google_Drive\Projet_Session_6-Controleur_Ecran\Gestion\Carte%20de%20bord%20Pascal-Emmanuel%20Lachance%202.docx#_Toc8667738)

[Figure 14 - Premier test du port parallèle sur STM (donnée : 0xA5) 21](#_Toc8667739)

[Figure 15 - Solder Paster Reflow Profile 30](#_Toc8667740)

[Figure 16 - C'est Noël! 30](#_Toc8667741)

[Figure 17 - Guidelines de création d'une *specific deviation* (MISRA-C:2004) 32](#_Toc8667742)

# Semaine 9

Cette semaine est toute mélangée en terme d’horaires.

Le lundi est un lundi, le mardi est un vendredi, le mercredi est un mardi, le jeudi est un mercredi et le vendredi est un jeudi.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lundi | Mardi | Mercredi | Jeudi | Vendredi |
| Lundi | Vendredi | Mardi | Mercredi | Jeudi |

### Objectifs

### Résultats

## Vendredi, 17 Mai 2019 (Jeudi)

### Enlever les condensateurs

### Enlever l’ampli op

### Se brancher sur l’alimentation de l’autre carte

## Jeudi, 16 Mai 2019 (Mercredi)

### Lignes striées

### Deuxième carte montée

## Mercredi, 15 Mai 2019 (Mardi)

### Affichage de quelques pixels

### Circuit d’alimentation remonté

### Commande de nouvelles pièces pour V1.1

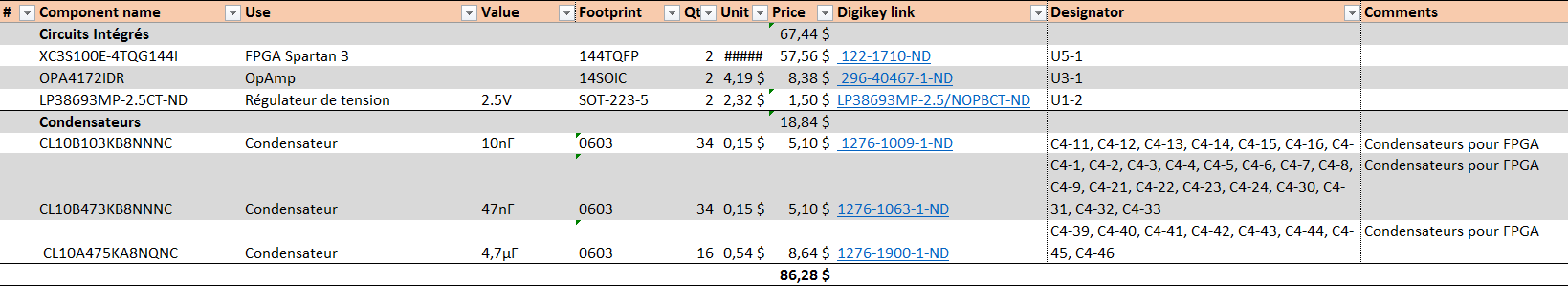


Figure - Deuxième commande de pièces

## Mardi, 14 Mai 2019 (Vendredi)

### Montage d’une deuxième carte

### Réécriture Data\_Ctrl

## Lundi, 13 Mai 2019

### Nous n’avions pas compris le principe d’un diviseur de tension

La tension de 25V que nous envoyons directement à l’écran sert également de source pour les autres niveaux du diviseur de tension nous permettant d’avoir 24V, 23V, 2V et 1V. Nous utilisions un ampli op pour isoler ces niveaux afin qu’ils ne s’influencent pas les uns et les autres.

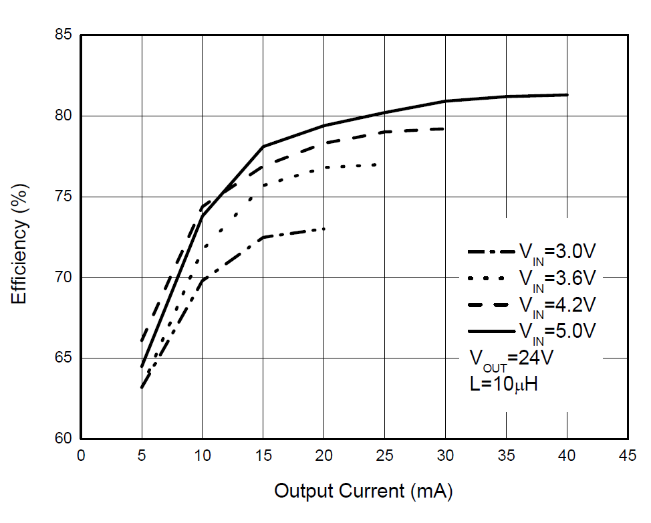
Cependant, le 25V n’était pas isolé et lorsqu’une charge y est branché, comme par exemple l’écran LCD, tous les niveaux de tension droppent parce que la charge devient le nouveau chemin le plus court.

Dans ce cas, le patch est également relativement facile à implémenter; il suffit de remplacer V5 (le dernier niveau de tension, qui n’a aucun besoin d’être isolé), par V0 (le niveau 25V servant de source à tous les autres niveaux de tension et qui est le plus important à isoler et protéger).

### Pas besoin de changer le boost converter!

En réalité, notre boost converter, AP3012, est correct. C’est notre ampli opérationnel qui pose problème, car il n’est pas très rail-to-rail. Nous montons le niveau de sortie du AP3012 à 28.2V, en mettant une résistance de 110kΩ au lieu de 100kΩ, ce qui nous permet d’alimenter l’ampli op à 28.2V, tout en continuant d’avoir une tension V0 de 25V grâce au potentiomètre, et donc de garder tous les niveaux de tension vers l’écran les mêmes. Simple fix.

### Changement du boost converter

Nous avions un convertisseur DC-DC prenant notre 5V et le transformant en 25V. Ce dernier, le AP3012, était rated pour 36V @ 500mA, ce qui était amplement suffisant pour nos besoins. Malheureusement, un coup d’œil plus profond dans sa datasheet nous aurait révélé que la valeur de 500mA indiquée était fausse! Nous ne pouvons sortir que à peu près 40mA.

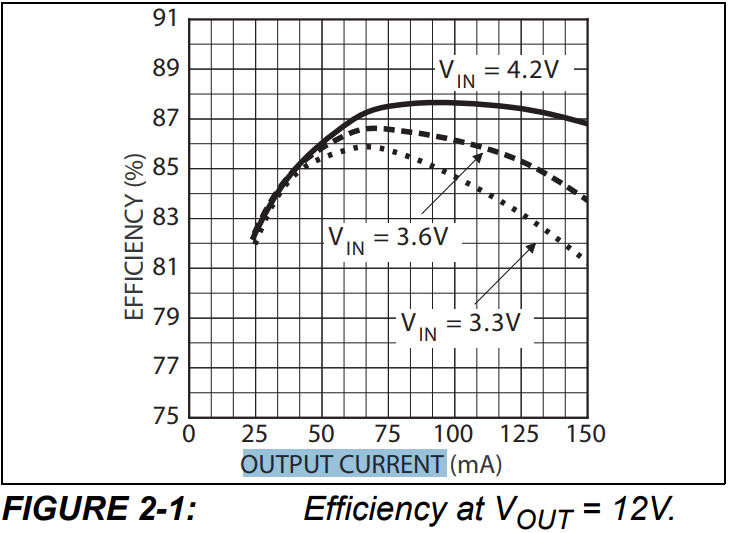
Nous trouvons heureusement un remplacement, le MIC2288YD5 de Microchip, un DC-DC boost converter, rated pour 34V @ 1A. Il ne va bien entendu pas être capable de sortir un tel courant, mais son courant de sortie réel est bien plus grand que celui du AP3012. Un avantage du MIC2288 est qu’il est complètement pin compatible avec le AP3012, donc on n’a rien à changer au niveau du schéma. Quant aux valeurs des composantes l’entourant, la diode Shottky que nous avons fera l’affaire, la valeur de bobine recommandée est exactement la même, à 10µH, et tout ce que nous aurions à changer serait la valeur des résistances R2-2 et R2-3

Figure 2 - Courant de sortie du régulateur AP3012

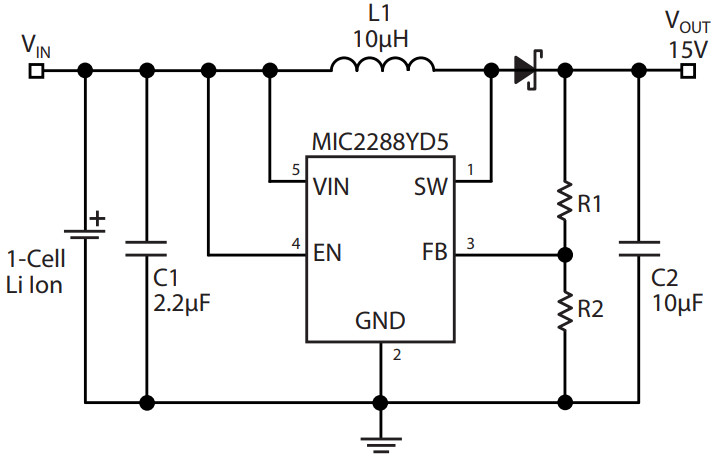
La valeur de tension Vout est donnée par l’équation suivante :

Figure 3 - Typical application circuit MIC2288

Figure 4 - Courant de sortie du MIC2288

Nous cherchons une valeur de sortie Vout de 25V, avec une valeur d’entrée Vref de 5V :

# Semaine 8

### Objectifs

Tester et souder notre PCB, puis le retester.

Afficher le chat (de la semaine 2) avec notre propre PCB.

Faire fonctionner le port parallèle avec écran et microcontrôleur.

Réviser le UART pour la version 1.1.

Tester les fonctions UART et I²C en VHDL.

Ajouter des fonctions graphiques à la librairie C.

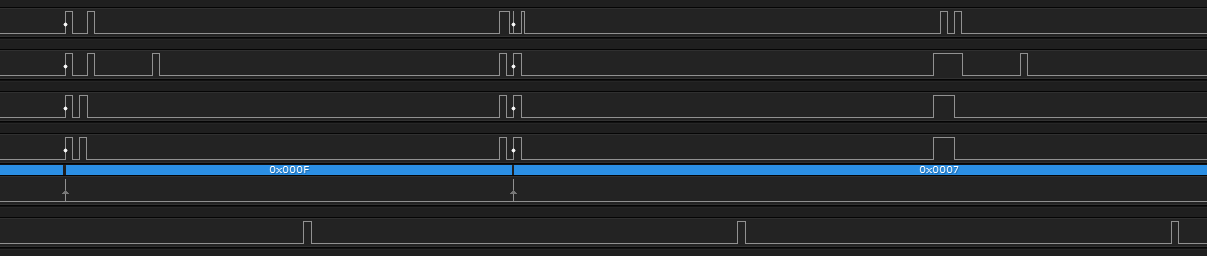
### Résultats

## Vendredi, 10 Mai 2019

### Resoudure FPGA

### Resoudure connecteur écran

### Problèmes de timing avec signal SHIFT



### Short sur l’alimentation de l’écran

## Jeudi, 9 Mai 2019

### Test tensions diviseur

### 2 patchs

### 1 patch?

### Condensateurs 0603 pour version 1.1

### Tests de l’écran avec chat

#### Alimentation shutdown

#### Data à 200mV

## Mercredi, 8 Mai 2019

### Correction short

Nous demandons de l’aide à Alain Champagne, chirurgien du fer à souder, afin de nous sauver de notre short entre le 1.2V et le GND sous le FPGA.

### Premier test!

### Inspections niveaux de tension

Les niveaux de tension sur la carte sont très beaux et stables, en partie probablement grâce aux très nombreux condensateurs présents sur la carte. Nous ne pouvons malheureusement pas encore tester le 25V et les tensions du diviseur de tension, car ces dernières sont contrôlées par le signal SHDN, provenant du FPGA.

### Xilinx ISE sur Windows 7 32-bits

### Programmation de notre carte par JTAG!

### Module SPI C & VHDL

## Mardi, 7 Mai 2019

### Inspection

#### Short sur 1.2V

### Fin de la soudure, bottom layer

## Lundi, 6 Mai 2019

### Soudure

#### Mise dans le four

Nous utilisons le profil #4 du four en classe. Les petites composantes sont bien soudées, mais le FPGA a quelques petits ponts. Il y avait également un pont sur le FTDI, et quelques petits condensateurs 0402 qui se sont soudés ensemble. Ces erreurs ont étés corrigées manuellement.

#### Mauvais padstacks condensateurs 4.7µF

Les condensateurs C1-4 et C1-13 doivent être 4.7µF en 0603. Nous avons des footprints 0603, mais nos composantes sont 1206. Heureusement, Yves a dans sa boîte de pièces des condensateurs 4.7µF format 0603 que nous pouvons emprunter.

#### Footprint switchés

Les footprints pour C2-3 et C2-2 sont switchés. Heureusement, les deux sont en parallèle, donc ça ne change rien.

# Semaine 7

### Objectifs

Nous devons faire fonctionner le port parallèle, objectif de la semaine dernière.

Nous voulons pouvoir afficher avec le UART afin de le tester.

Nous voulons tester nos fonctions I2C; autant du côté VHDL que C, rien n’a été testé à date.

Avec un peu de chance, nous recevrons nos PCBs cette semaine.

### Résultats

Le port parallèle du côté VHDL fonctionne tout seul, il reste à le tester avec un écran et un microcontrôleur.

Les fonctions UART et I²C sont fonctionnelles dans le C. Des tests standalones ont étés effectués avec le UART en VHDL, mais rien encore en I²C.

Nous avons reçu nos PCB et nous sommes prêts à les souder dès lundi prochain!

## Dimanche, 5 Mai 2019

### Plus d’ajustements sur le boîtier

Figure 5 - Ajustements sur trous de vis

J’essaye de placer toutes les composantes through-hole sur le PCB. Malheureusement, nos trous de vis surélevés sont dans le chemin, il me faut donc ajuster ces derniers.

### Test du port parallèle VHDL

Le port Paralllèle en VHDL fonctionne de façon standalone, et théoriquement avec le module Data\_Ctrl, il serait également fonctionnel avec notre trame de communication. Il reste à tester cette fonctionalité.

## Samedi, 4 Mai 2019

### Ajustements sur boîtier

Tous les trous de vis étaient décalés de 0.7mm dans des directions différentes. J’ajuste également la position du connecteur USB et du connecteur Parallèle, lance l’impression et retourne me reposer chez moi.

(Un accident de badminton me pousse à avoir 5 points de suture au visage, je prends donc mon samedi plus relax.)

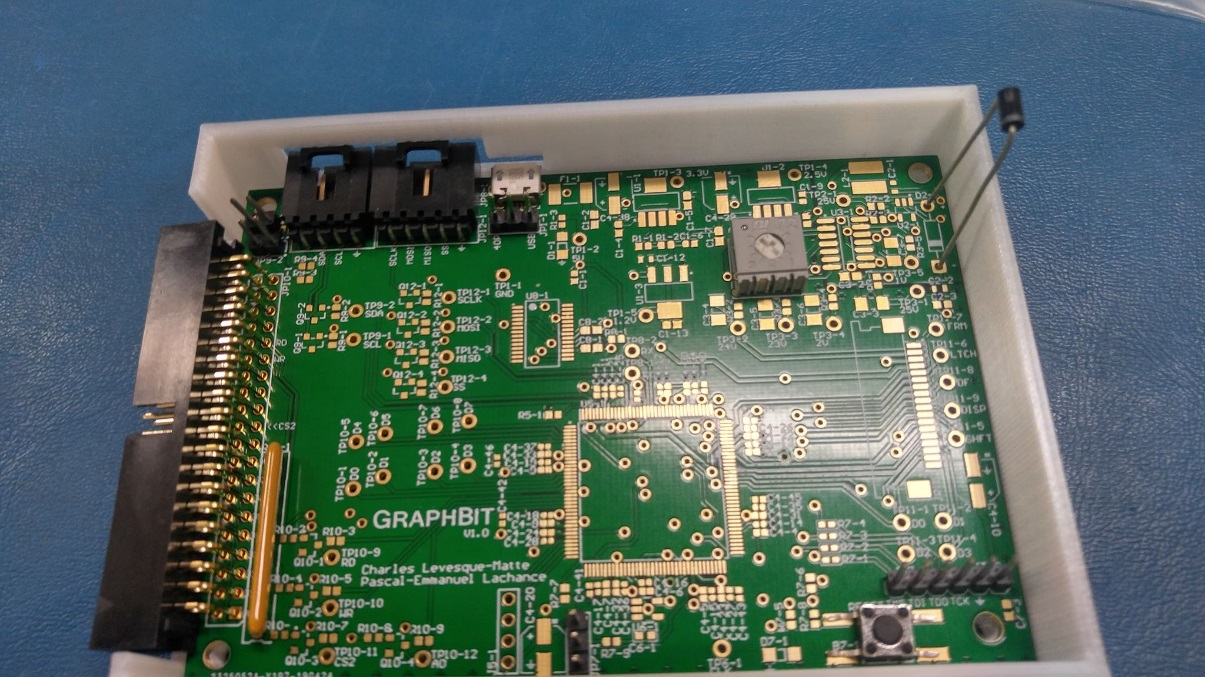


Figure 6 - Boîtier avec composantes through-hole installés sur le PCB.

## Vendredi, 3 Mai 2019

### Tests UART

Charles réussi à faire fonctionner le UART aussi du premier coup dans la librairie en C! Miracle!

Il ne reste que le SPI, des fonctions graphiques, et la traduction de la librairie STM vers une librairie Dallas!

### Réception de nos PCBs!

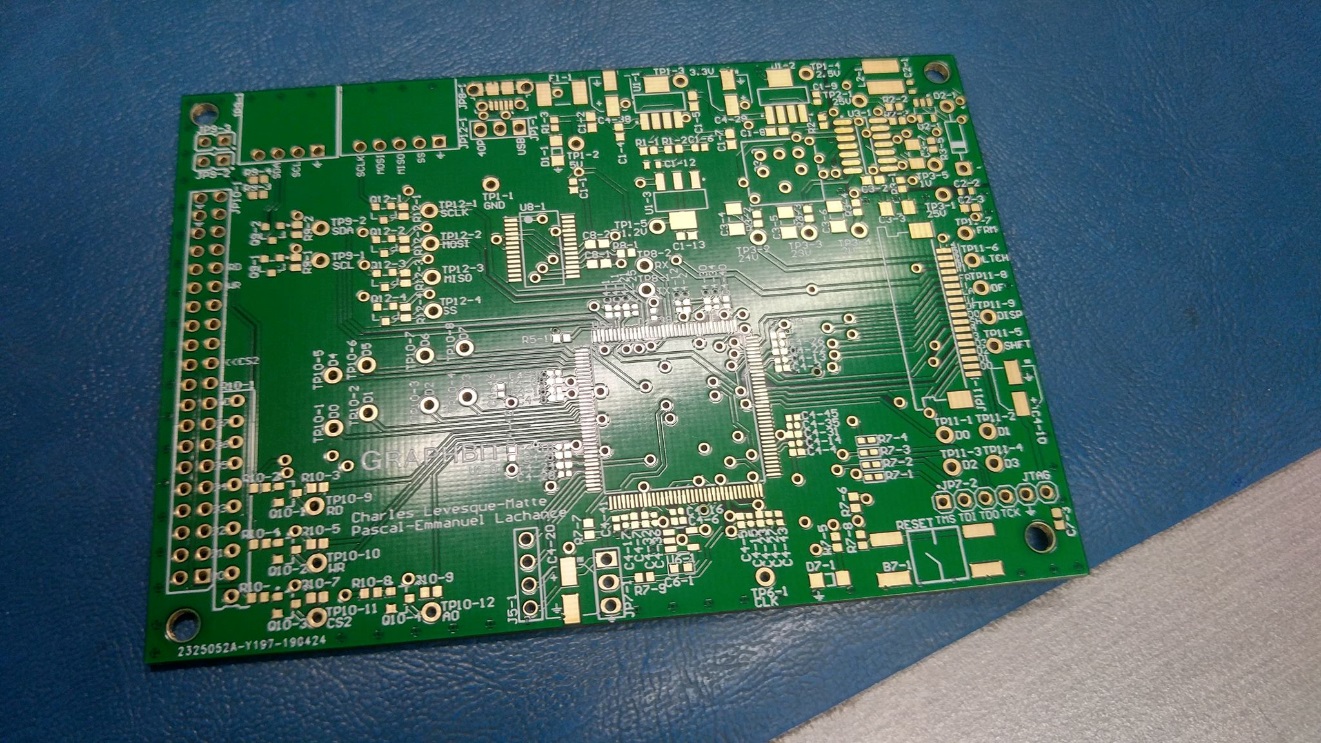


Figure 7 - PCB reçu, top face

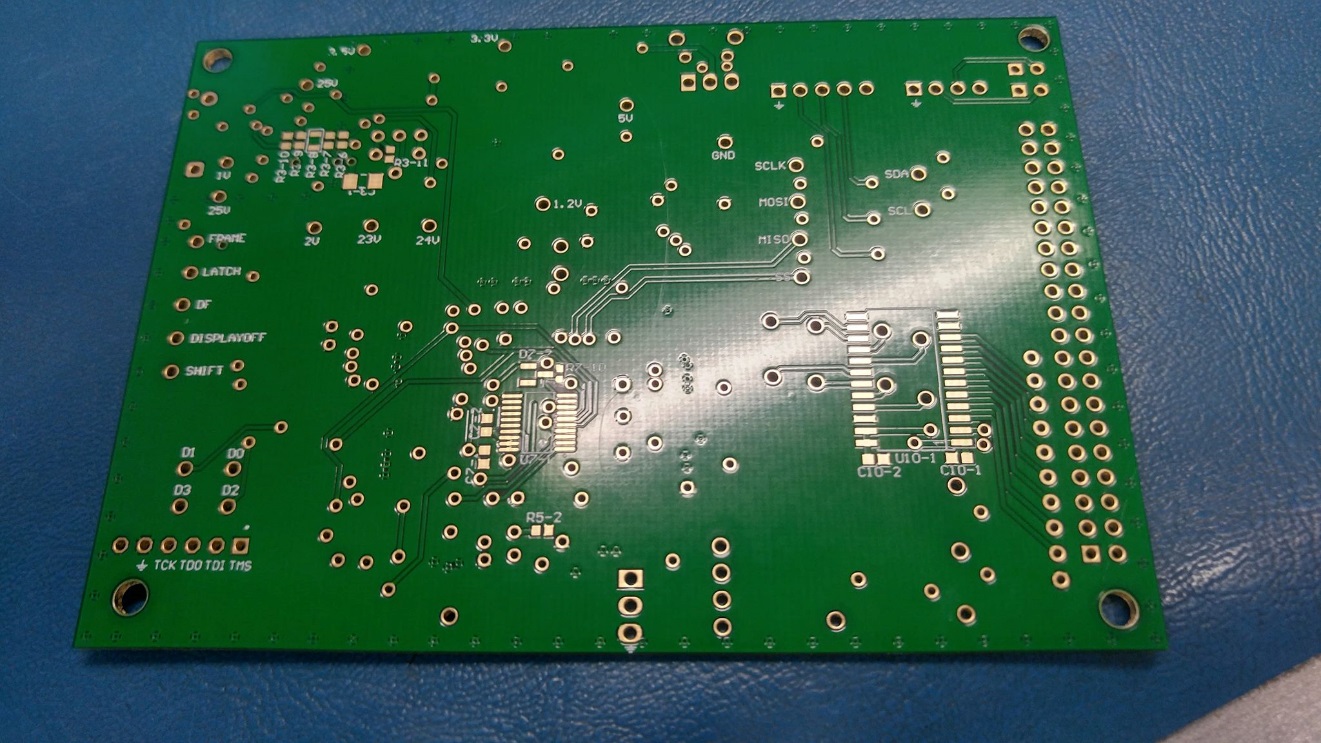


Figure 8 - PCB reçu, bottom face

### Nouveau logo



Figure 9 - Nouveau logo GraphBit

## Jeudi, 2 Mai 2019

### Switch codeurs

Suite aux tests de I²C, moi et Charles changeons de tâche!

Charles, travaillant précédemment sur le code VHDL, vient travailler sur la librairie en C, et je prends le code VHDL en main.

Charles s’étant informé sur les guidelines de la MISRA et étant lui aussi un bon programmeur, je suis confiant qu’il pourra continuer à travailler sur la librairie C sans problèmes et que ses ajouts seront bons et sécuritaires.

Pour ma part, je comprends pourquoi Charles sacrait tant devant son code VHDL depuis les dernières semaines.

### Tests I2C

Miracle! Notre I²C fonctionne lui aussi de premier coup, comme le port parallèle!

Nous avions eu initialement des inquiétudes, car le code ne semblait pas fonctionner, mais c’était parce que nous n’avions pas de dispositif connecté au bus I²C, donc nous ne recevions pas de *Ack*. Nous avons donc branché un dispositif bidon (un VL160X Satel de ST, un capteur de distance I²C), et écrit à l’adresse de ce dernier pour qu’il nous renvoie des *Ack*, et tout fonctionne à merveille!

#### LCD\_Write\_I2C

static LCD\_ERROR\_T LCD\_Write\_I2C(const LCD\_COMMAND\_T Command, const uint8\_t pData[], const size\_t dataSize)

{

LCD\_ERROR\_T I2C\_Error = NO\_ERROR;

if (LCD\_Struct.State == LCD\_READY)

{

LCD\_Struct.State = LCD\_BUSY;

HAL\_StatusTypeDef I2C\_State = HAL\_OK;

/\* Check if dataSize is too big \*/

if ((dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD) > dataSize)

{

/\* Initialize data structure \*/

/\* (Variable length array are illegal in C89, make sure the compiler used is based on C99) \*/

size\_t write\_i = 0x00U;

uint8\_t Data[(dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD)];

uint8\_t Checksum = (dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD);

Checksum += Command;

/\* Dynamic stack memory allocation, we must check if it's valid \*/

if ((&Data[0] != NULL) && (&Data[0] != 0))

{

size\_t temp\_dataSize = (dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD);

/\* The amount of data is the first and second bytes sent \*/

Data[0] = (uint8\_t)((temp\_dataSize & 0xFF00U) >> 8U);

Data[1] = (uint8\_t)(temp\_dataSize & 0x00FFU);

/\* Command is the second byte sent \*/

Data[2] = Command;

/\* Check pointer value \*/

/\* It is possible for pData to be a NULL pointer, but only if dataSize is equal to 0 \*/

if ((pData != NULL) || ((pData == NULL) && (dataSize != 0)))

{

if (pData != NULL)

{

/\* Populating the array with pData \*/

for (write\_i = (LCD\_I2C\_SENDBUFFER\_SIZEADD - 0x01U); write\_i <= (dataSize + (LCD\_I2C\_SENDBUFFER\_SIZEADD - 0x02U)); write\_i++)

{

Data[write\_i] = pData[(write\_i - (LCD\_I2C\_SENDBUFFER\_SIZEADD - 0x01U))];

Checksum += Data[write\_i];

}

}

/\* Calculating checksum's 2's complement value and placing it at the end of the array \*/

Checksum = (~Checksum) + 0x01U;

Data[dataSize + (LCD\_I2C\_SENDBUFFER\_SIZEADD - 0x01U)] = Checksum;

/\* Calculating a timeout value \*/

uint32\_t timeoutValue = 0x03U \* + (uint32\_t)(dataSize / 0x04U);

/\* Sending data via I2C \*/

I2C\_State = HAL\_I2C\_Master\_Transmit(LCD\_Struct.hi2c, LCD\_I2C\_ADRESS, &Data[0], (uint16\_t)(dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD), timeoutValue);

if (I2C\_State == HAL\_OK)

{

/\* Operations with the module have been completed successfully, reset state \*/

LCD\_Struct.State = LCD\_READY;

}

else /\* I2C\_State != HAL\_OK \*/

{

/\* There was a HAL error \*/

switch (I2C\_State)

{

case HAL\_BUSY:

I2C\_Error = E\_I2C\_HAL\_BUSY;

break;

case HAL\_TIMEOUT:

I2C\_Error = E\_I2C\_HAL\_TIMEOUT;

break;

case HAL\_ERROR:

/\* Nested switch case, to check which error arose \*/

switch (LCD\_Struct.hi2c->ErrorCode)

{

case HAL\_I2C\_ERROR\_AF: /\* Acknowledge failure \*/

/\* This error is the one most likely to occur in HAL\_I2C\_Master\_Transmit \*/

I2C\_Error = E\_I2C\_HAL\_ERROR\_AF;

break;

case HAL\_I2C\_ERROR\_BERR: /\* Bus error \*/

I2C\_Error = E\_I2C\_HAL\_ERROR\_BERR;

break;

case HAL\_I2C\_ERROR\_ARLO: /\* Arbitration lost \*/

I2C\_Error = E\_I2C\_HAL\_ERROR\_ARLO;

break;

case HAL\_I2C\_ERROR\_OVR: /\* Overrun \*/

I2C\_Error = E\_I2C\_HAL\_ERROR\_OVR;

break;

case HAL\_I2C\_ERROR\_DMA: /\* DMA \*/

I2C\_Error = E\_I2C\_HAL\_ERROR\_DMA;

break;

default: /\* Timeout error or magic \*/

/\* This shouldn't ever happen \*/

I2C\_Error = E\_I2C\_HAL\_ERROR;

break;

}

break;

default:

I2C\_Error = E\_I2C\_HAL\_ERROR; /\* This shouldn't ever happen \*/

break;

}

}

}

else /\* (pData == NULL) && (dataSize != 0) \*/

{

/\* This functions accepts null pointers to a data buffer

\* But only if the datasize is equal to 0, indicating that

\* we don't want to send any data beyond the command & checksum

\* If the pointer is null but dataSize has a value, there was an error \*/

I2C\_Error = E\_WRITE\_ARRAY;

assert(pData != NULL);

}

}

else /\* (&Data[0] == NULL) || (&Data[0] == 0) \*/

{

/\* Array pointer is invalid \*/

I2C\_Error = E\_WRITE\_ARRAY;

assert(&Data[0] != NULL);

assert(&Data[0] != 0);

}

}

else /\* (dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD) > dataSize\*/

{

/\* The data exceeds the data size limit (65535 - LCD\_I2C\_SENDBUFFER\_SIZEADD)\*/

/\* This if instruction may seem weird. But in the case of an overflow, the

\* result of the addition would be lower than the initial value of dataSize \*/

I2C\_Error = E\_WRITE\_SIZE;

}

}

else /\* LCD\_Struct.State != LCD\_READY \*/

{

/\* The module isn't ready to write \*/

switch (LCD\_Struct.State) /\* Check its state \*/

{

case LCD\_BUSY:

/\* Module is currently performing operations, try later \*/

I2C\_Error = E\_I2C\_WRITE\_S\_BUSY;

break;

case LCD\_FAILURE:

/\* Module has critically failed \*/

I2C\_Error = E\_I2C\_WRITE\_S\_FAILURE;

break;

default:

/\* You should never get here \*/

LCD\_Struct.State = LCD\_FAILURE;

I2C\_Error = E\_I2C\_WRITE\_S\_FAILURE;

break;

}

}

return (I2C\_Error);

}

### Mercredi, 1er Mai 2019

### UART pour version 1.1

Nous avons fait une belle bévue.

Nous n’avons pas de vrai UART sur notre carte. En effet, bien que nous avons un FTDI pour convertir du USB en UART, et que tout ça semble être beau, nous n’avons pas de connecteur nous permettant de connecter une autre carte par UART. Cela veut dire que nous pouvons nous interfacer avec un ordinateur facilement, mais que si l’on souhaite communiquer par UART avec une autre carte, il faut soit convertir le UART avec une autre carte (telle que la carte Prolific du magasin), puis brancher ce signal UART->USB dans notre propre carte qui fait USB->UART, soit, dans le cas de la STM, se taper la stack USB, qui est toute une autre histoire (ça serait un très très gros bonus, apparemment que des équipes dans le passé ont essayé de se faire du USB et y ont passés la session.

Au lieu de faire cela, nous nous rajoutons un connecteur pour la version 1.1.

Avec ce connecteur vient un ajustement de tension et des circuits de protections avec des diodes (afin de ne pas avoir de conflits entre le connecteur et le FTDI).

Il va rester à faire des ajustements à ce circuit, car les signaux Rx et Tx sont inversés par le circuit d’ajustement de tension, et que les diodes descendent trop le niveau de tension (-0.7V sur un signal 3.3V)

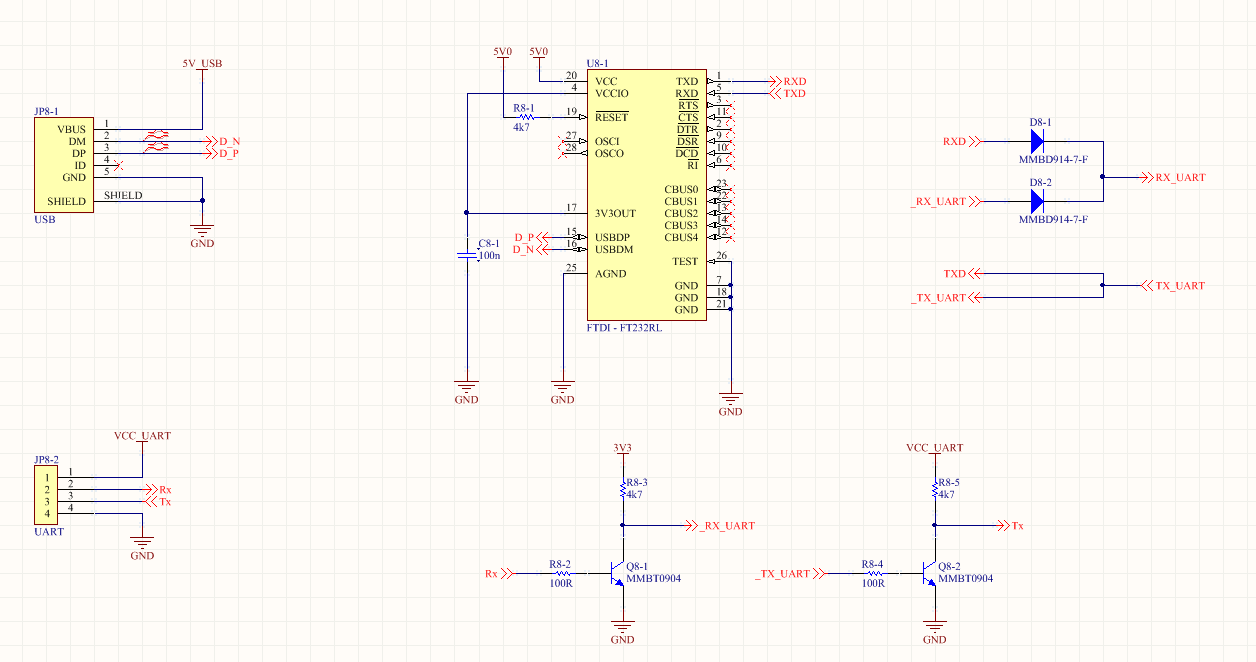


Figure 10 - Rajout d'un connecteur pour le UART

## Mardi, 30 avril 2019

### Modèle 3D du boîtier

En faisant notre PCB, nous avons fait des modifications aux emplacements prédéterminés des vis et des connecteurs, donc j’applique ces modifications au modèle 3D, augmente la marge de manœuvre pour la taille, et enlève le overhang du connecteur 40-pins.

### Nos PCBs arrivent jeudi!

Figure 11 - Modèle 3D ajusté

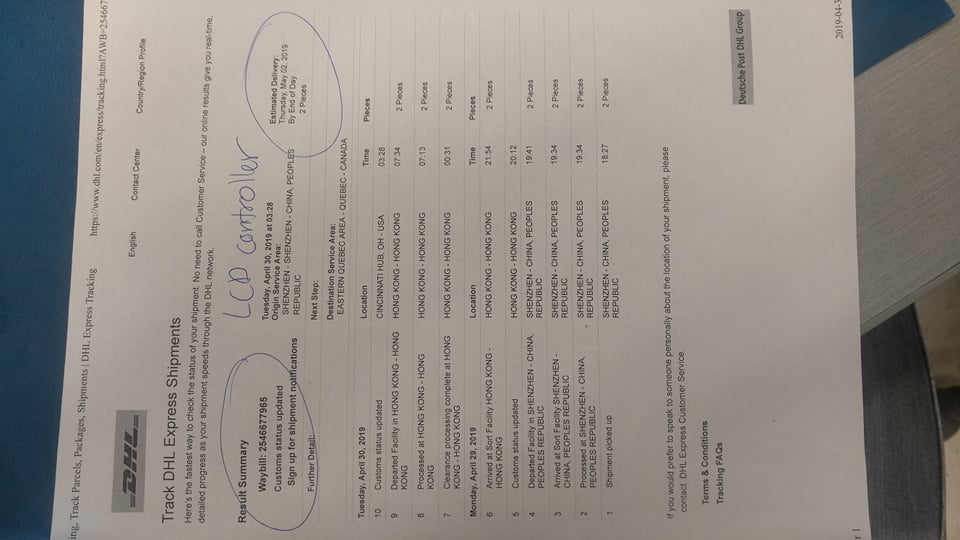
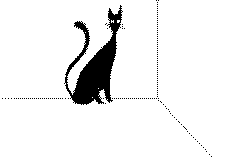
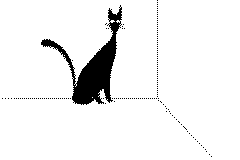
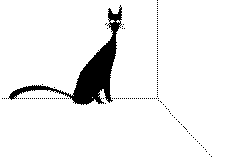


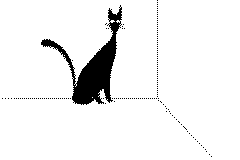
Figure 12 - Shipment tracking

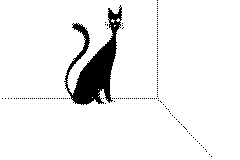
### Chat envoyé par port parallèle

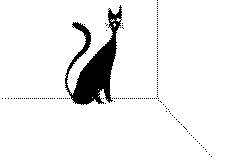












# Semaine 6

### Objectifs

Nous voulons cette semaine faire fonctionner le port parallèle sur STM et sur le VHDL. Afficher juste du texte serait bien, mais une image complète serait l’idéal!

### Résultats

Le code Parallèle sur VHDL fonctionne en théorie, mais d’autres partie du code font en sorte que le module de communication parallèle n’est même pas généré par ISE. De ce côté, notre objectif échoue.

Cependant, notre librairie C réussi à faire bien fonctionner le port parallèle! Du premier coup en plus!

On reçoit également nos composantes.

## Dimanche, 28 avril 2019

### Entête de fonction

Je fais un petit entête de fonction standard pour la librairie LCD et commence à la remplir pour toutes les fonctions

#### Entête de LCD\_Init\_I2C

/\* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . \*/

/\* LCD\_Init\_I2C

\* LCD\_ERROR\_T LCD\_Init\_I2C(I2C\_HANDLER\* hi2c, TIM\_HANDLER\* htim)

\* Author : Pascal-Emmanuel Lachance

\* Date : 2019-04-20

\*

\* Initializes the LCD module in I2C mode.

\*

\* Called by : none

\* Function calls :

\* - LCD\_Config\_Timer()

\* - LCD\_Clear()

\*

\* Parameters : I2C\_HANDLER\* hi2c, TIM\_HANDLER\* htim

\* - hi2c : pointer to a I2C\_HandleTypeDef structure from the HAL librairies

\* - htim : pointer to a TIM\_HandleTypeDef structure from the HAL librairies

\* Returns: LCD\_ERROR\_T Init\_I2C\_Error

\* - E\_I2C\_INIT : If the I2C\_HANDLER hi2c pointer is invalid

\* - E\_TIM\_HAL\_ERROR : Error during HAL initialization of the timer

\* - E\_TIM\_HAL\_ERROR\_CLKSOURCE : Error during HAL initialization of the timer clocksource

\* - E\_TIM\_HAL\_ERROR\_SLAVE : Error during HAL initialization of the timer slave

\* - E\_TIM\_HAL\_ERROR\_MASTER : Error during HAL initialization of the timer master

\* - E\_TIM\_INIT : Invalid TIM\_HANDLER htim pointer

\* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . \*/

## Samedi, 27 avril 2019

### Ajustements port parallèle

Finalement le code ne marchait pas complètement du premier coup, il y avait quelques problèmes avec le calcul et l’écriture du checksum, mais tout est maintenant réglé.

## Vendredi, 26 avril 2019

### Port parallèle dans la librairie LCD

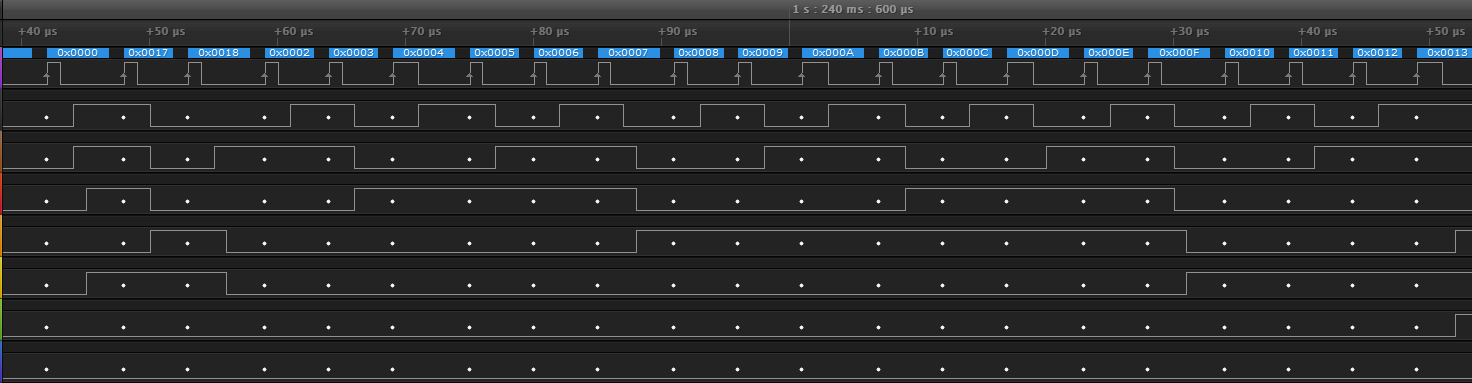
Encore d’autre code qui fonctionne du premier coup!

Figure 13 - Premier test du port parallèle

#### LCD\_Write\_Parallel

/\* MISRA-C : 2004 prohibits the multiple break statement within a loop (Rule 14.6)

\* We're creating a specific deviation for this function, LCD\_Write\_Parallel, that can exit the data transfer loop

\* if an error arises either while transmitting data on the parallel bus.

\* This is done in order to immediatly handle an error in the loop if it arises, and is more secure than not having two break statements. \*/

static LCD\_ERROR\_T LCD\_Write\_Parallel(const uint8\_t Command, const uint8\_t pData[], const size\_t dataSize)

{

LCD\_ERROR\_T P\_Error = NO\_ERROR;

if ((Command == LCD\_PARALLEL\_COMMAND\_TEXT) || (Command == LCD\_PARALLEL\_COMMAND\_GRAPH))

{

if (LCD\_Struct.State == LCD\_READY)

{

LCD\_Struct.State = LCD\_BUSY;

/\* Check if dataSize is too big \*/

if ((dataSize + LCD\_PARALLEL\_SENDBUFFER\_SIZEADD) > dataSize)

{

/\* Initialize data structure \*/

/\* (Variable lenghts array are illegal in C89, make sure the compiler used is based on C99) \*/

size\_t write\_i = 0x00U;

uint8\_t Data[(dataSize + LCD\_PARALLEL\_SENDBUFFER\_SIZEADD)];

uint8\_t Checksum = (dataSize + LCD\_PARALLEL\_SENDBUFFER\_SIZEADD);

/\* Dynamic stack memory allocation, we must check if it's valid \*/

if ((&Data[0] != NULL) && (&Data[0] != 0))

{

memset(Data, 0x0000, (dataSize + LCD\_PARALLEL\_SENDBUFFER\_SIZEADD));

size\_t temp\_dataSize = (dataSize + LCD\_PARALLEL\_SENDBUFFER\_SIZEADD);

/\* The amount of data is the first and second bytes sent \*/

Data[0] = (uint8\_t)((temp\_dataSize & 0xFF00U) >> 8U);

Data[1] = (uint8\_t)(temp\_dataSize & 0x00FFU);

/\* Command is the second byte sent \*/

Data[2] = Command;

/\* Check pointer value \*/

/\* It is possible for pData to be a NULL pointer, but only if dataSize is equal to 0 \*/

if (((pData != NULL) && (dataSize != 0)))

{

/\* Populating the array with pData \*/

for (write\_i = (LCD\_PARALLEL\_SENDBUFFER\_SIZEADD - 0x01U); write\_i <= (dataSize + (LCD\_PARALLEL\_SENDBUFFER\_SIZEADD - 0x01U)); write\_i++)

{

Data[write\_i] = pData[(write\_i - 0x01U)];

Checksum += Data[write\_i];

}

/\* Calculating checksum's 2's complement value and placing it at the end of the array \*/

Checksum = (~Checksum) + 0x01U;

Data[(LCD\_PARALLEL\_SENDBUFFER\_SIZEADD - 0x01U)] = Checksum;

/\* Calculating a timeout value \*/

uint32\_t timeoutValue = 0x03U \* + (uint32\_t)(dataSize / 0x04U);

P\_ERROR\_T Parallel\_Error = P\_NO\_ERROR;

/\* Sending data on the parallel bus \*/

for (write\_i = 0x00U; write\_i < (dataSize + LCD\_PARALLEL\_SENDBUFFER\_SIZEADD); write\_i++)

{

/\* Send a byte \*/

Parallel\_Error = Parallel\_Write(Data[write\_i], (Command & 0x01U), CS2);

if (Parallel\_Error != P\_NO\_ERROR)

{

/\* Immediatly exit loop if there's an error \*/

P\_Error = E\_PARALLEL;

assert(Parallel\_Error == P\_NO\_ERROR);

break;

}

Parallel\_Error = Parallel\_CS(CS2, PIN\_LOW);

if (Parallel\_Error != P\_NO\_ERROR)

{

/\* Immediatly exit loop if there's an error \*/

P\_Error = E\_PARALLEL;

assert(Parallel\_Error == P\_NO\_ERROR);

break;

}

}

}

else /\* (pData == NULL) && (dataSize != 0) \*/

{

/\* This functions accepts null pointers to a data buffer

\* But only if the datasize is equal to 0, indicating that

\* we don't want to send any data beyond the command & checksum

\* If the pointer is null but dataSize has a value, there was an error \*/

P\_Error = E\_WRITE\_ARRAY;

assert(pData != NULL);

}

}

else /\* (&Data[0] == NULL) || (&Data[0] == 0) \*/

{

/\* Array pointer is invalid \*/

P\_Error = E\_WRITE\_ARRAY;

assert(&Data[0] != NULL);

assert(&Data[0] != 0);

}

}

else /\* (dataSize + LCD\_I2C\_SENDBUFFER\_SIZEADD) > dataSize\*/

{

/\* The data exceeds the data size limit (65535 - LCD\_I2C\_SENDBUFFER\_SIZEADD)\*/

/\* This if instruction may seem weird. But in the case of an overflow, the

\* result of the addition would be lower than the initial value of dataSize \*/

P\_Error = E\_WRITE\_SIZE;

}

}

else /\* LCD\_Struct.State != LCD\_READY \*/

{

/\* The module isn't ready to write \*/

switch (LCD\_Struct.State) /\* Check its state \*/

{

case LCD\_BUSY:

/\* Module is currently performing operations, try later \*/

P\_Error = E\_PARALLEL\_WRITE\_S\_BUSY;

break;

case LCD\_FAILURE:

/\* Module has critically failed \*/

P\_Error = E\_PARALLEL\_WRITE\_S\_FAILURE;

break;

default:

/\* You should never get here \*/

LCD\_Struct.State = LCD\_FAILURE;

P\_Error = E\_PARALLEL\_WRITE\_S\_FAILURE;

break;

}

}

}

else /\* (Command != LCD\_PARALLEL\_COMMAND\_TEXT) && (Command !=

CD\_PARALLEL\_COMMAND\_GRAPH) \*/

{ /\* Command isn't known \*/

P\_Error = E\_PARALLEL\_WRITE\_COMMAND;

}

return (P\_Error);

}

## Jeudi, 25 avril 2019

### Test du port parallèle

Je réussis aujourd’hui (et hier) à écrire un port parallèle complet, très flexible et versatile, qui marche presque du premier coup!

Il reste quelques fonctions à écrire, notamment pour lire le port, et pour le réinitialiser (pour libérer les GPIOs), mais il est déjà fonctionnel!

Son seul défaut est qu’il n’est pas réellement parallèle, il écrit les données de façon séquentielle sur les GPIOs de la STM, il y a donc un délai entre l’écriture de D0 et celle de D7.

Autrement, la librairie accepte un nombre variable de pins pour données, adresses et chip select, qui sont simplement configurées en donnant au code une liste des pins que l’on souhaite assigner.

Figure 14 - Exemple de configuration du port parallèle

*Parallel\_InitData(8, GPIOE, GPIO\_PIN\_8, GPIOE, GPIO\_PIN\_9, GPIOE, GPIO\_PIN\_10, GPIOE, GPIO\_PIN\_11, GPIOE, GPIO\_PIN\_12, GPIOE, GPIO\_PIN\_13, GPIOE, GPIO\_PIN\_14, GPIOE, GPIO\_PIN\_15);*

*Parallel\_InitAdress(5, GPIOD, GPIO\_PIN\_8, GPIOD, GPIO\_PIN\_9, GPIOD, GPIO\_PIN\_10, GPIOD, GPIO\_PIN\_11, GPIOB, GPIO\_PIN\_1);*

*Parallel\_InitChipSelect(4, GPIOB, GPIO\_PIN\_11, GPIOB, GPIO\_PIN\_12, GPIOB, GPIO\_PIN\_13, GPIOB, GPIO\_PIN\_14);*

*Parallel\_Init();*

*Parallel\_Write(0xA5, 0x01, 2);*

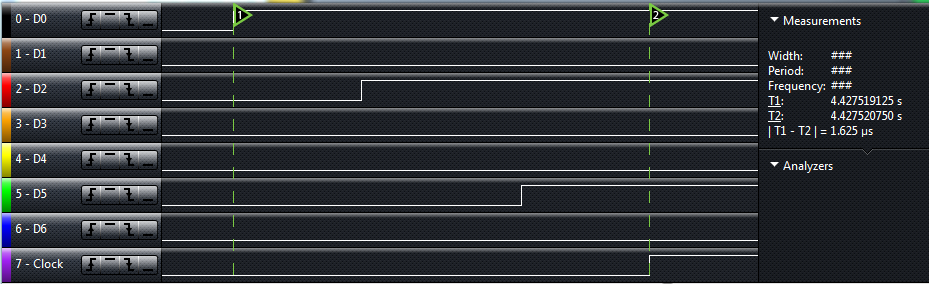


Figure 15 - Premier test du port parallèle sur STM (donnée : 0xA5)

#### Parallel.h

#ifndef PARALLEL\_H

#define PARALLEL\_H

/\* --------------------------------------------------------------------------------------------- \*/

/\* Includes \*/

#include <assert.h> /\* For the assert() macro of the standard \*/

#include <stdbool.h> /\* For boolean type declaration \*/

#include <stdint.h> /\* For standard integer type redeclaration \*/

#include <stdarg.h> /\* For variadic functions \*/

#include "stm32f4xx\_hal.h" /\* For HAL API peripherals and functions \*/

/\* --------------------------------------------------------------------------------------------- \*/

/\* Definitions \*/

#define P\_DATA\_READY 0x01U

#define P\_ADRESS\_READY 0x02U

#define P\_CS\_READY 0x04U

#define P\_ALL\_READY 0xFFU

/\* --------------------------------------------------------------------------------------------- \*/

/\* Enumerated types \*/

typedef enum

{

P\_NO\_ERROR = 0, /\* Default state \*/

P\_E\_NONE = 0, /\* Default state \*/

P\_E\_INVALID\_NUM, /\* Invalid value of argument num indicating number of pins \*/

P\_E\_INVALID\_PORT, /\* Invalid port pointer to GPIO\_TypeDef \*/

P\_E\_INVALID\_PIN, /\* Invalid pin number \*/

P\_E\_NOBUS, /\* None of the bus has been initialized before module init \*/

P\_E\_DATABUS, /\* The data bus hasn't been initialized before module init \*/

P\_E\_ADRESSBUS, /\* The adress bus hasn't been initialized before module init \*/

P\_E\_CSBUS, /\* The Chip Select bus hasn't been initialized before module init \*/

P\_E\_NOTREADY, /\* Attempt to write on the bus before initialization was complete \*/

P\_E\_INVALID\_CS, /\* Attempt to use a chip select that hasn't been initialized \*/

}P\_ERROR\_T;

/\* --------------------------------------------------------------------------------------------- \*/

/\* Structures \*/

typedef struct

{

GPIO\_TypeDef\* Port;

uint32\_t Pin;

bool used;

}GPIO\_T;

typedef struct

{

GPIO\_T Data[8];

GPIO\_T Adress[16];

GPIO\_T ChipSelect[6];

uint8\_t ready;

}PARALLEL\_PORT\_T;

/\* --------------------------------------------------------------------------------------------- \*/

/\* Functions \*/

P\_ERROR\_T Parallel\_InitData(size\_t num, ...);

P\_ERROR\_T Parallel\_InitAdress(size\_t num, ...);

P\_ERROR\_T Parallel\_InitChipSelect(size\_t num, ...);

P\_ERROR\_T Parallel\_Init(void);

P\_ERROR\_T Parallel\_Write(uint8\_t Data, uint16\_t Adress, uint8\_t ChipSelect);

#endif

#### Parallel.c

#include "Parallel.h"

/\* --------------------------------------------------------------------------------------------- \*/

/\* Create an initialize the Parallel structure \*/

PARALLEL\_PORT\_T Parallel\_Struct =

{

{ /\* Data bus \*/

{ NULL, 0x00, false }, /\* D0 \*/

{ NULL, 0x00, false }, /\* D1 \*/

{ NULL, 0x00, false }, /\* D2 \*/

{ NULL, 0x00, false }, /\* D3 \*/

{ NULL, 0x00, false }, /\* D4 \*/

{ NULL, 0x00, false }, /\* D5 \*/

{ NULL, 0x00, false }, /\* D6 \*/

{ NULL, 0x00, false }, /\* D7 \*/

},

{ /\* Adress bus \*/

{ NULL, 0x00, false}, /\* A0 \*/

{ NULL, 0x00, false}, /\* A1 \*/

{ NULL, 0x00, false}, /\* A2 \*/

{ NULL, 0x00, false}, /\* A3 \*/

{ NULL, 0x00, false}, /\* A4 \*/

{ NULL, 0x00, false}, /\* A5 \*/

{ NULL, 0x00, false}, /\* A6 \*/

{ NULL, 0x00, false}, /\* A7 \*/

{ NULL, 0x00, false}, /\* A8 \*/

{ NULL, 0x00, false}, /\* A9 \*/

{ NULL, 0x00, false}, /\* A10 \*/

{ NULL, 0x00, false}, /\* A11 \*/

{ NULL, 0x00, false}, /\* A12 \*/

{ NULL, 0x00, false}, /\* A13 \*/

{ NULL, 0x00, false}, /\* A14 \*/

{ NULL, 0x00, false}, /\* A15 \*/

},

{ /\* Chip Select \*/

{ NULL, 0x00, false }, /\* CS0 \*/

{ NULL, 0x00, false }, /\* CS1 \*/

{ NULL, 0x00, false }, /\* CS2 \*/

{ NULL, 0x00, false }, /\* CS3 \*/

{ NULL, 0x00, false }, /\* CS4 \*/

{ NULL, 0x00, false }, /\* CS5 \*/

},

0x00U /\* uint8\_t ready \*/

};

/\* --------------------------------------------------------------------------------------------- \*/

/\* Private Function Declarations \*/

void Parallel\_ResetData(void);

void Parallel\_ResetAdress(void);

void Parallel\_ResetChipSelect(void);

void Parallel\_WriteIO(GPIO\_T\* GPIO, bool state);

/\* --------------------------------------------------------------------------------------------- \*/

/\* Public Function Definitions \*/

/\* --------------------------------------------------------------------------------------------- \*/

/\* Initialization functions \*/

/\* The following 3 functions are non-compliant with MISRA-C:2004 guidelines, breaking rule 16.1

\* "Functions shall not be defined with a variable number of arguments"

\* Specific deviations have been raised for all 3 of the functions and security measures have been

\* taken seriously.

\* Noncompliant functions:

\* P\_ERROR\_T Parallel\_InitData(size\_t num, ...);

\* P\_ERROR\_T Parallel\_InitAdress(size\_t num, ...);

\* P\_ERROR\_T Parallel\_InitChipSelect(size\_t num, ...);

\*

\* Prohibited functions of <stdarg.h> used:

\* void va\_start (va\_list ap, paramN); - Used in Parallel\_InitData & Parallel\_InitAdress & Parallel\_InitChipSelect

\* type va\_arg (va\_list ap, type); - Used in Parallel\_InitData & Parallel\_InitAdress & Parallel\_InitChipSelect

\* void va\_end (va\_list ap); - Used in Parallel\_InitData & Parallel\_InitAdress & Parallel\_InitChipSelect

\*\*/

/\* MISRA-C : 2004 prohibits the usage of variadic functions (Rule 16.1)

\* We're creating a specific deviation for this function, Parallel\_InitData, that takes as input

\* a variable amount of parameters.

\* This is done in order to make the librairy easier to use and more flexible.

\* We're checking every value after casting them to the proper type, and making sure all are valid before use.

\* Any doubt of invalidity instantly aborts the input process.

\* In the case of an error during data acquisition, all values are reset and an error flag is raised

\* In the case of an error during data acquisition, function va\_end is called to stop the acquisition and free memory \*/

P\_ERROR\_T Parallel\_InitData(size\_t num, ...)

{

P\_ERROR\_T InitData\_Error = P\_NO\_ERROR;

va\_list ap;

/\* Check if number of argument is correct \*/

if ((num <= 8) && (num > 0))

{

uint8\_t Data\_variadic\_i;

/\* Initialize va\_list for num number of argument \*/

va\_start(ap, num);

/\* Read every argument passed \*/

for (Data\_variadic\_i = 0x00; Data\_variadic\_i < num; Data\_variadic\_i++)

{

/\* Retrieve Port \*/

Parallel\_Struct.Data[Data\_variadic\_i].Port = (GPIO\_TypeDef\*) va\_arg(ap, GPIO\_TypeDef\*);

/\* Immediatly check if port is valid \*/

if ((Parallel\_Struct.Data[Data\_variadic\_i].Port == NULL) || (Parallel\_Struct.Data[Data\_variadic\_i].Port == 0))

{

/\* Port isn't valid! \*/

InitData\_Error = P\_E\_INVALID\_PORT;

assert(Parallel\_Struct.Data[Data\_variadic\_i].Port != NULL);

assert(Parallel\_Struct.Data[Data\_variadic\_i].Port != 0);

va\_end(ap); /\* End the variable list \*/

break; /\* Exit the loop \*/

}

/\* Retrieve Pin \*/

Parallel\_Struct.Data[Data\_variadic\_i].Pin = (uint32\_t) va\_arg(ap, uint32\_t);

/\* Immediatly check if pin is valid \*/

if ((Parallel\_Struct.Data[Data\_variadic\_i].Pin >= 0x8000U) && (Parallel\_Struct.Data[Data\_variadic\_i].Pin < 0U))

{ /\* If the pin number is bigger than 15 or smaller than 0 \*/

/\* Pin number isn't valid \*/

InitData\_Error = P\_E\_INVALID\_PIN;

assert(Parallel\_Struct.Data[Data\_variadic\_i].Pin <= 15U);

assert(Parallel\_Struct.Data[Data\_variadic\_i].Pin >= 0U);

va\_end(ap); /\* End the variable list \*/

break; /\* Exit the loop \*/

}

Parallel\_Struct.Data[Data\_variadic\_i].used = true;

}

/\* Check if an error was encountered within the loop \*/

if (InitData\_Error == P\_NO\_ERROR)

{

/\* End the variadic list and clean memory \*/

va\_end(ap);

Parallel\_Struct.ready |= P\_DATA\_READY;

}

else /\* InitData\_Error != P\_NO\_ERROR \*/

{ /\* There was an error during the loop \*/

/\* Reset all values in the data structure \*/

Parallel\_ResetData();

}

}

else /\* (num > 8) || (num < 0) \*/

{ /\* number of argument invalid \*/

InitData\_Error = P\_E\_INVALID\_NUM;

assert((num <= 8) && (num > 0));

}

return (InitData\_Error);

}

/\* MISRA-C : 2004 prohibits the usage of variadic functions (Rule 16.1)

\* We're creating a specific deviation for this function, Parallel\_InitAdress, that takes as input

\* a variable amount of parameters.

\* This is done in order to make the librairy easier to use and more flexible.

\* We're checking every value after casting them to the proper type, and making sure all are valid before use.

\* Any doubt of invalidity instantly aborts the input process.

\* In the case of an error during data acquisition, all values are reset and an error flag is raised

\* In the case of an error during data acquisition, function va\_end is called to stop the acquisition and free memory \*/

P\_ERROR\_T Parallel\_InitAdress(size\_t num, ...)

{

P\_ERROR\_T InitAdress\_Error = P\_NO\_ERROR;

va\_list ap;

/\* Check if number of argument is correct \*/

if ((num <= 16) && (num > 0))

{

uint8\_t Adress\_variadic\_i;

/\* Initialize va\_list for num number of argument \*/

va\_start(ap, num);

/\* Read every argument passed \*/

for (Adress\_variadic\_i = 0x00; Adress\_variadic\_i < num; Adress\_variadic\_i++)

{

/\* Retrieve Port \*/

Parallel\_Struct.Adress[Adress\_variadic\_i].Port = (GPIO\_TypeDef\*) va\_arg(ap, GPIO\_TypeDef\*);

/\* Immediatly check if port is valid \*/

if ((Parallel\_Struct.Adress[Adress\_variadic\_i].Port == NULL) || (Parallel\_Struct.Adress[Adress\_variadic\_i].Port == 0))

{

/\* Port isn't valid! \*/

InitAdress\_Error = P\_E\_INVALID\_PORT;

assert(Parallel\_Struct.Adress[Adress\_variadic\_i].Port != NULL);

assert(Parallel\_Struct.Adress[Adress\_variadic\_i].Port != 0);

va\_end(ap); /\* End the variable list \*/

break; /\* Exit the loop \*/

}

/\* Retrieve Pin \*/

Parallel\_Struct.Adress[Adress\_variadic\_i].Pin = (uint32\_t) va\_arg(ap, uint32\_t);

/\* Immediatly check if pin is valid \*/

if ((Parallel\_Struct.Adress[Adress\_variadic\_i].Pin >= 0x8000U) && (Parallel\_Struct.Adress[Adress\_variadic\_i].Pin < 0U))

{

/\* If the pin number is bigger than 15 or smaller than 0 \*/

/\* Pin number isn't valid \*/

InitAdress\_Error = P\_E\_INVALID\_PIN;

assert(Parallel\_Struct.Adress[Adress\_variadic\_i].Pin <= 15U);

assert(Parallel\_Struct.Adress[Adress\_variadic\_i].Pin >= 0U);

va\_end(ap); /\* End the variable list \*/

break; /\* Exit the loop \*/

}

Parallel\_Struct.Adress[Adress\_variadic\_i].used = true;

}

/\* Check if an error was encountered within the loop \*/

if (InitAdress\_Error == P\_NO\_ERROR)

{

/\* End the variadic list and clean memory \*/

va\_end(ap);

Parallel\_Struct.ready |= P\_ADRESS\_READY;

}

else /\* InitAdress\_Error = P\_NO\_ERROR \*/

{ /\* There was an error in the loop \*/

/\* Reset all values in the adress structure \*/

Parallel\_ResetAdress();

}

}

else /\* (num > 8) || (num < 0) \*/

{

/\* number of argument invalid \*/

InitAdress\_Error = P\_E\_INVALID\_NUM;

assert((num <= 8) && (num > 0));

}

return (InitAdress\_Error);

}

/\* MISRA-C : 2004 prohibits the usage of variadic functions (Rule 16.1)

\* We're creating a specific deviation for this function, Parallel\_InitChipSelect, that takes as input

\* a variable amount of parameters.

\* This is done in order to make the librairy easier to use and more flexible.

\* We're checking every value after casting them to the proper type, and making sure all are valid before use.

\* Any doubt of invalidity instantly aborts the input process.

\* In the case of an error during data acquisition, all values are reset and an error flag is raised

\* In the case of an error during data acquisition, function va\_end is called to stop the acquisition and free memory \*/

P\_ERROR\_T Parallel\_InitChipSelect(size\_t num, ...)

{

P\_ERROR\_T InitChipSelect\_Error = P\_NO\_ERROR;

va\_list ap;

/\* Check if number of argument is correct \*/

if ((num <= 6) && (num > 0))

{

uint8\_t ChipSelect\_variadic\_i;

/\* Initialize va\_list for num number of argument \*/

va\_start(ap, num);

/\* Read every argument passed \*/

for (ChipSelect\_variadic\_i = 0x00; ChipSelect\_variadic\_i < num; ChipSelect\_variadic\_i++)

{

/\* Retrieve Port \*/

Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Port = (GPIO\_TypeDef\*) va\_arg(ap, GPIO\_TypeDef\*);

/\* Immediatly check if port is valid \*/

if ((Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Port == NULL) || (Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Port == 0))

{

/\* Port isn't valid! \*/

InitChipSelect\_Error = P\_E\_INVALID\_PORT;

assert(Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Port != NULL);

assert(Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Port != 0);

va\_end(ap); /\* End the variable list \*/

break; /\* Exit the loop \*/

}

/\* Retrieve Pin \*/

Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Pin = (uint32\_t) va\_arg(ap, uint32\_t);

/\* Immediatly check if pin is valid \*/

if ((Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Pin >= 0x8000U) && (Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Pin < 0U))

{

/\* If the pin number is bigger than 15 or smaller than 0 \*/

/\* Pin number isn't valid \*/

InitChipSelect\_Error = P\_E\_INVALID\_PIN;

assert(Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Pin <= 15U);

assert(Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].Pin >= 0U);

va\_end(ap); /\* End the variable list \*/

break; /\* Exit the loop \*/

}

Parallel\_Struct.ChipSelect[ChipSelect\_variadic\_i].used = true;

}

/\* Check if an error was encountered within the loop \*/

if (InitChipSelect\_Error == P\_NO\_ERROR)

{

/\* End the variadic list and clean memory \*/

va\_end(ap);

Parallel\_Struct.ready |= P\_CS\_READY;

}

else /\* InitChipSelect\_Error != P\_NO\_ERROR \*/

{

/\* There was an error in the loop \*/

/\* Reset all values in the chip select structure \*/

Parallel\_ResetChipSelect();

}

}

else if (num == 0)

{ /\* We don't want to initialize a chip select \*/

Parallel\_Struct.ready |= P\_CS\_READY;

}

else /\* (num > 8) || (num < 0) \*/

{ /\* number of argument invalid \*/

InitChipSelect\_Error = P\_E\_INVALID\_NUM;

assert((num <= 8) && (num > 0));

}

return (InitChipSelect\_Error);

}

P\_ERROR\_T Parallel\_Init(void)

{

P\_ERROR\_T P\_Init\_Error = P\_NO\_ERROR;

if (Parallel\_Struct.ready == ((P\_DATA\_READY) | (P\_ADRESS\_READY) | (P\_CS\_READY)))

{

/\* HAL Initialization structure for the GPIOs \*/

GPIO\_InitTypeDef GPIO\_InitStruct;

uint8\_t P\_Init\_i;

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOE\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOH\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();

/\* Configure GPIO pin Output Level for Data bus \*/

for (P\_Init\_i = 0; P\_Init\_i < (sizeof(Parallel\_Struct.Data) / sizeof(GPIO\_T)); P\_Init\_i++)

{

if (Parallel\_Struct.Data[P\_Init\_i].used == true)

{

/\* Put pin in low state \*/

Parallel\_WriteIO(&Parallel\_Struct.Data[P\_Init\_i], 0);

}

}

/\* Configure GPIO pin Output Level for Adress bus \*/

for (P\_Init\_i = 0; P\_Init\_i < (sizeof(Parallel\_Struct.Adress) / sizeof(GPIO\_T)); P\_Init\_i++)

{

if (Parallel\_Struct.Adress[P\_Init\_i].used == true)

{

/\* Put pin in low state \*/

Parallel\_WriteIO(&Parallel\_Struct.Adress[P\_Init\_i], 0);

}

}

/\* Configure GPIO pin Output Level for Chip Selects \*/

for (P\_Init\_i = 0; P\_Init\_i < (sizeof(Parallel\_Struct.ChipSelect) / sizeof(GPIO\_T)); P\_Init\_i++)

{

if (Parallel\_Struct.ChipSelect[P\_Init\_i].used == true)

{

/\* Put pin in low state \*/

Parallel\_WriteIO(&Parallel\_Struct.ChipSelect[P\_Init\_i], 0);

}

}

/\* Configure Data bus GPIO pins \*/

for (P\_Init\_i = 0; P\_Init\_i < (sizeof(Parallel\_Struct.Data) / sizeof(GPIO\_T)); P\_Init\_i++)

{

if (Parallel\_Struct.Data[P\_Init\_i].used == true)

{

GPIO\_InitStruct.Pin = Parallel\_Struct.Data[P\_Init\_i].Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_MEDIUM;

HAL\_GPIO\_Init(Parallel\_Struct.Data[P\_Init\_i].Port, &GPIO\_InitStruct);

}

}

/\* Configure Adress bus GPIO pins \*/

for (P\_Init\_i = 0; P\_Init\_i < (sizeof(Parallel\_Struct.Adress) / sizeof(GPIO\_T)); P\_Init\_i++)

{

if (Parallel\_Struct.Adress[P\_Init\_i].used == true)

{

GPIO\_InitStruct.Pin = Parallel\_Struct.Adress[P\_Init\_i].Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_MEDIUM;

HAL\_GPIO\_Init(Parallel\_Struct.Adress[P\_Init\_i].Port, &GPIO\_InitStruct);

}

}

/\* Configure Chip Select bus GPIO pins \*/

for (P\_Init\_i = 0; P\_Init\_i < (sizeof(Parallel\_Struct.ChipSelect) / sizeof(GPIO\_T)); P\_Init\_i++)

{

if (Parallel\_Struct.ChipSelect[P\_Init\_i].used == true)

{

GPIO\_InitStruct.Pin = Parallel\_Struct.ChipSelect[P\_Init\_i].Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_MEDIUM;

HAL\_GPIO\_Init(Parallel\_Struct.ChipSelect[P\_Init\_i].Port, &GPIO\_InitStruct);

}

}

Parallel\_Struct.ready = P\_ALL\_READY;

}

else /\* Parallel\_Struct.ready != ((P\_DATA\_READY) | (P\_ADRESS\_READY) | (P\_CS\_READY)) \*/

{ /\* One of the bus hasn't been intialized properly before this function call \*/

/\* The functions Parallel\_InitData, Parallel\_InitAdress and Parallel\_InitChipSelect need to be called before this function \*/

if (Parallel\_Struct.ready == 0x00)

{ /\* None of the bus has been initiaized \*/

P\_Init\_Error = P\_E\_NOBUS;

}

if ((Parallel\_Struct.ready & P\_DATA\_READY) != P\_DATA\_READY)

{

/\* The data bus hasn't been initialized \*/

P\_Init\_Error = P\_E\_DATABUS;

}

if ((Parallel\_Struct.ready & P\_ADRESS\_READY) != P\_ADRESS\_READY)

{

/\* The adress bus hasn't been initialized \*/

P\_Init\_Error = P\_E\_ADRESSBUS;

}

if ((Parallel\_Struct.ready & P\_CS\_READY) != P\_CS\_READY)

{

P\_Init\_Error = P\_E\_CSBUS;

}

}

return (P\_Init\_Error);

}

/\* --------------------------------------------------------------------------------------------- \*/

/\* Low-level functions definitions \*/

void Parallel\_WriteIO(GPIO\_T\* GPIO, bool state)

{

HAL\_GPIO\_WritePin(GPIO->Port, GPIO->Pin, (GPIO\_PinState) state);

}

/\* --------------------------------------------------------------------------------------------- \*/

/\* Mid-level functions definitions \*/

P\_ERROR\_T Parallel\_Write(uint8\_t Data, uint16\_t Adress, uint8\_t ChipSelect)

{

P\_ERROR\_T ParallelWrite\_Error = P\_NO\_ERROR;

/\* Check if module is ready \*/

if (Parallel\_Struct.ready == 0xFFU)

{

uint8\_t ParallelWrite\_i;

/\* Write Data on the data bus \*/

for (ParallelWrite\_i = 0x00U; ParallelWrite\_i < (sizeof(Parallel\_Struct.Data) / sizeof(GPIO\_T)); ParallelWrite\_i++)

{

if (Parallel\_Struct.Data[ParallelWrite\_i].used == true)

{

Parallel\_WriteIO(&Parallel\_Struct.Data[ParallelWrite\_i], ((Data >> ParallelWrite\_i) & 0x01U));

}

else /\* We've reached bits that aren't used \*/

{

break; /\* Exit loop \*/

}

}

/\* Write Adress on the adress bus \*/

for (ParallelWrite\_i = 0x00U; ParallelWrite\_i < (sizeof(Parallel\_Struct.Adress) / sizeof(GPIO\_T)); ParallelWrite\_i++)

{

if (Parallel\_Struct.Adress[ParallelWrite\_i].used == true)

{

Parallel\_WriteIO(&Parallel\_Struct.Adress[ParallelWrite\_i], ((Adress >> ParallelWrite\_i) & 0x01U));

}

else /\* We've reached bits that aren't used \*/

{

break; /\* Exit loop \*/

}

}

/\* Drive the chip select pin high and disable other chip select pins \*/

if (Parallel\_Struct.ChipSelect[ChipSelect].used == true)

{ /\* The chosen chip select exists \*/

for (ParallelWrite\_i = 0x00U; ParallelWrite\_i < (sizeof(Parallel\_Struct.ChipSelect) / sizeof(GPIO\_T)); ParallelWrite\_i++)

{

if (Parallel\_Struct.ChipSelect[ParallelWrite\_i].used == true)

{

(ParallelWrite\_i == ChipSelect) ? Parallel\_WriteIO(&Parallel\_Struct.ChipSelect[ParallelWrite\_i], 1U) : Parallel\_WriteIO(&Parallel\_Struct.ChipSelect[ParallelWrite\_i], 0U);

}

else /\* We've reached bits that aren't used \*/

{

break; /\* Exit loop \*/

}

}

}

else /\* Parallel\_Struct.ChipSelect[ChipSelect].used != true \*/

{ /\* Chosen Chip Select isn't available \*/

ParallelWrite\_Error = P\_E\_INVALID\_CS;

}

}

else /\* Parallel\_Struct.ready != 0xFF \*/

{ /\* The module hasn't been initialized properly \*/

ParallelWrite\_Error = P\_E\_NOTREADY;

}

return (ParallelWrite\_Error);

}

/\* --------------------------------------------------------------------------------------------- \*/

/\* Private function definition \*/

void Parallel\_ResetData(void)

{

uint8\_t DataReset\_i;

for (DataReset\_i = 0; DataReset\_i < sizeof(Parallel\_Struct.Data); DataReset\_i++)

{

Parallel\_Struct.Data[DataReset\_i].Port = NULL;

Parallel\_Struct.Data[DataReset\_i].Pin = 0x00U;

Parallel\_Struct.Data[DataReset\_i].used = false;

}

}

void Parallel\_ResetAdress(void)

{

uint8\_t AdressReset\_i;

for (AdressReset\_i = 0; AdressReset\_i < sizeof(Parallel\_Struct.Data); AdressReset\_i++)

{

Parallel\_Struct.Adress[AdressReset\_i].Port = NULL;

Parallel\_Struct.Adress[AdressReset\_i].Pin = 0x00U;

Parallel\_Struct.Adress[AdressReset\_i].used = false;

}

}

void Parallel\_ResetChipSelect(void)

{

uint8\_t CSReset\_i;

for (CSReset\_i = 0; CSReset\_i < sizeof(Parallel\_Struct.Data); CSReset\_i++)

{

Parallel\_Struct.ChipSelect[CSReset\_i].Port = NULL;

Parallel\_Struct.ChipSelect[CSReset\_i].Pin = 0x00U;

Parallel\_Struct.ChipSelect[CSReset\_i].used = false;

}

}

### Reflow Profile de notre solder paste

La solder paste [disponible au magasin](https://www.digikey.ca/product-detail/en/chip-quik-inc/SMDLTLFP/SMDLTLFP-ND/2682721) (vielle de 2 ans).

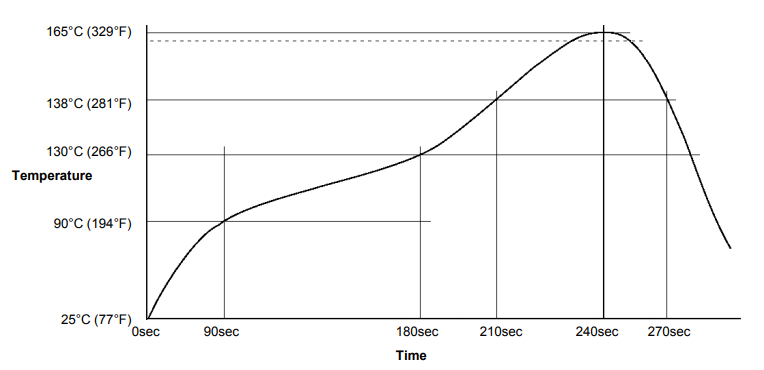


Figure 16 - Solder Paster Reflow Profile

### Réception de nos pièces!



Figure 17 - C'est Noël!

## Mercredi, 24 avril 2019

### Variadic functions

Une fonction variadique est une fonction capable d’accepter un nombre variable d’arguments. Comme la commande « printf » de la standard librairy. On utilise des fonctions variadic afin de faciliter le setup du port parallèle.

En effet, on veut pouvoir permettre à l’utilisateur d’initialiser un port parallèle comme il le souhaite. Créer un bus de 4 bits ou de 8 bits ou de 3 bits. Créer 1 ligne d’adresse ou 8 ou 16 ou 13. Plus de liberté et de flexibilité sont donné à l’utilisateur de cette façon, et la librairie parallèle est plus générale.

Cependant, nous devons pour cela créer une déviation spécifique à la règle 16.1 de la MISRA-C :2004 (« *Functions shall not be defined with a variable number of arguments* »), avec les informations de déviation contenues dans les entêtes des 3 fonctions variadiques.

### Port parallèle sur STM

#### Data :

* D0 : PE8
* D1 : PE9
* D2 : PE10
* D3 : PE11
* D4 : PE12
* D5 : PE13
* D6 : PE14
* D7 : PE15

#### Adress

* A0 : PD8
* A1 : PD9
* A2 : PD10
* A3 : PD11
* A4 : PB1

#### ChipSelect

* CS0 : PB11
* CS1 : PB12
* CS2 : PB13
* CS3 : PB14

## Mardi, 23 avril 2019

### Commande des pièces

### Allocation de mémoire dynamique

Afin d’optimiser la mémoire, le temps d’opération et surtout le temps de communication, lorsque nous devons employer un tableau pour faire des calculs, au lieu de garder un seul immense tableau toujours en mémoire, correspondant à l’écran et à ses milliers de pixels, nous créons temporairement un tableau, faisons nos calculs dans ce dernier, puis supprimons ce tableau afin de libérer la mémoire allouée.

Se faisant, nous créons une déviation spécifique à la règle 20.4 de la MISRA-C :2004 (« *Dynamic Heap Memory Allocation shall not but used* »), ces informations sont contenues dans les entêtes des 4 fonctions dérogeant à cette règle, qui encapsulent la création et destruction d’espace mémoire dynamique.

Nous sommes toujours MISRA-C :2004 compliant, car nous avons levés une *specific deviation* et répondu aux critères de cette dernière.

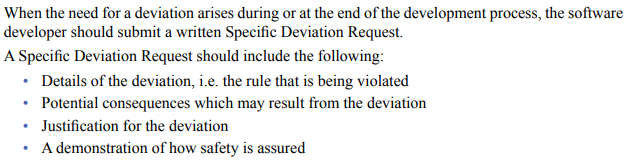


Figure 18 - Guidelines de création d'une *specific deviation* (MISRA-C:2004)

### Port parallèle STM32

Bien que la carte STM32 dispose de périphériques pour le I²C, UART et SPI, elle manque un port parallèle.

Alain Champagne a écrit auparavant une librairie permettant de simuler un port parallèle au travers des GPIOs de la carte STM, et c’est cette librairie qui est utilisée pour faire fonctionner l’écran. Cependant, elle n’est définitivement pas MISRA-C :2004 compliant, et quelques améliorations et modifications pourraient y être apportées.

Heureusement, il existe une application note de ST, [AN4666](https://www.st.com/content/ccc/resource/technical/document/application_note/7a/88/df/e3/d3/36/40/29/DM00169730.pdf/files/DM00169730.pdf/jcr:content/translations/en.DM00169730.pdf), *Parallel synchronous transmission using GPIO and DMA*. Un code est également disponible pour une carte Discovery STM32F429 (très similaire à notre Discovery STM32F407).

Malheureusement, à part me donner l’idée d’employer la DMA, cette application note et cet exemple de code sont incompatibles avec notre librairie.

## Lundi, 22 avril 2019

### Librairie STM32

Je commence aujourd’hui la librairie STM32 pour l’écran LCD.

Cette librairie doit pouvoir communiquer avec l’écran au travers des interfaces suivantes :

* Interface Parallèle
* Interface I2C
* Interface SPI
* Interface UART

Nous nous donnons également le défi de rendre nos librairies [MISRA-C :2004](http://caxapa.ru/thumbs/468328/misra-c-2004.pdf) compliant, c’est-à-dire de suivre les règles établies par le document et d’écrire un code sécuritaire, voire « student-proof ».

Je commence donc la création de la section I2C de la librairie.

Nous voulons rendre cette dernière très facile à utiliser, donc après initialisation, toutes les fonctions qui peuvent être appelées sont en fait des pointeurs de fonction, qui se font rediriger à la bonne fonction après initialisation (par exemple LCD\_Write, qui devient LCD\_Write\_I2C, ou LCD\_Write\_SPI).

### Plugin VisualGDB

VisualGDB est un plugin pour Visual Studio permettant de créer du code pour STM32 (entre autres) et de débugger pour cette famille de microcontrôleurs.

Bien que le plugin soit payant, une version d’évaluation (avec période de 30 jours) est offerte gratuitement. J’en fait l’essai et commence à développer la librairie STM32 dans l’environnement supérieur de Visual Studio 2017.