****

CHARLES LÉVESQUE-MATTE & Pascal-Emmanuel Lachance

Cégep Limoilou

Présenté à Daniel Côté

Hiver 2019

RAPPORT DE PROJET

Table des matières

[RAPPORT DE PROJET 0](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781522)

[Table des illustrations 3](#_Toc10781523)

[Description du projet 5](#_Toc10781524)

[Défi 5](#_Toc10781525)

[Kit étudiant 5](#_Toc10781526)

[Protections 5](#_Toc10781527)

[Conformité MISRA-C : 2004 6](#_Toc10781528)

[Description fonctionnelle du projet 7](#_Toc10781529)

[FPGA 7](#_Toc10781530)

[JTAG 7](#_Toc10781531)

[Écran 8](#_Toc10781532)

[Entrées 8](#_Toc10781533)

[USB 8](#_Toc10781534)

[Parallèle 9](#_Toc10781535)

[UART 9](#_Toc10781536)

[I²C 9](#_Toc10781537)

[SPI 10](#_Toc10781538)

[Trame de communication 10](#_Toc10781539)

[Modes 10](#_Toc10781540)

[Mode Graphique 10](#_Toc10781541)

[Mode Texte 10](#_Toc10781542)

[Librairie C 11](#_Toc10781543)

[LCD 11](#_Toc10781544)

[Exemple de code : 11](#_Toc10781545)

[Parallel 11](#_Toc10781546)

[Exemple de code : 11](#_Toc10781547)

[Logiciel C# 0](#_Toc10781548)

[Site Web 0](#_Toc10781549)

[Schéma bloc général 1](#_Toc10781550)

[Schéma blocs électriques 2](#_Toc10781551)

[Schéma général 2](#_Toc10781552)

[Entrées 3](#_Toc10781553)

[I2C & SPI 3](#_Toc10781554)

[Parallèle 3](#_Toc10781555)

[USB & UART 3](#_Toc10781556)

[Sorties 0](#_Toc10781557)

[Alimentation 0](#_Toc10781558)

[Schéma blocs VHDL 0](#_Toc10781559)

[Schémas électriques V1.0 0](#_Toc10781560)

[PCB V1.0 5](#_Toc10781561)

[Schémas V1.1 13](#_Toc10781562)

[Valeur de R2-2 13](#_Toc10781563)

[Footprint de C2-2 13](#_Toc10781564)

[Valeur de R3-0 14](#_Toc10781565)

[Changer les condensateurs 0402 par des 0603 14](#_Toc10781566)

[Connecter le signal M2 qui permet de charger la EEPROM 14](#_Toc10781567)

[Passer V0 par ampliop 15](#_Toc10781568)

[Connecter le signal CE\_PROM 16](#_Toc10781569)

[Ajout du UART 16](#_Toc10781570)

[Connecteur à l’envers! 17](#_Toc10781571)

[Code VHDL 0](#_Toc10781572)

[Librairie C 1](#_Toc10781573)

[Exemple et Schéma Bloc 1](#_Toc10781574)

[Liste de variables globales 3](#_Toc10781575)

[Conformité MISRA-C :2004 3](#_Toc10781576)

[Parallel 3](#_Toc10781577)

[LCD 4](#_Toc10781578)

[Programme C# 0](#_Toc10781579)

[Site Web 0](#_Toc10781580)

# Table des illustrations

[Figure 1 – PCB GraphBit V1.0 montrant fièrement son FPGA 6](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781581)

[Figure 2 - Disposition des éléments principaux sur le PCB GraphBit V1.0 7](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781582)

[Figure 3 - Coordonnées des pixels aux extrémités de l'écran 8](#_Toc10781583)

[Figure 4 - Schéma bloc USB 8](#_Toc10781584)

[Figure 5 - Connecteur USB 8](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781585)

[Figure 6 - Schéma bloc parallèle 9](#_Toc10781586)

[Figure 7 - Connecteur Parallèle 9](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781587)

[Figure 8 - Connecteur I2C 9](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781588)

[Figure 9 - Connecteur UART 9](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781589)

[Figure 10 - Connecteur SPI 10](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781590)

[Figure 11 - Mode graphique 10](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781591)

[Figure 12 - Mode graphique et texte mélangés 10](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781592)

[Figure 13 - Schéma bloc général 1](#_Toc10781593)

[Figure 14 - Schéma bloc I2C & SPI 3](#_Toc10781594)

[Figure 15 - Schéma bloc parallèle 3](#_Toc10781595)

[Figure 16 - Schéma bloc USB 3](#_Toc10781596)

[Figure 17 - Schéma bloc connecteur écran 0](#_Toc10781597)

[Figure 18 - Schéma bloc alimentation 0](#_Toc10781598)

[Figure 19 - Schéma bloc VHDL simplifié 0](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781599)

[Figure 20 - Schéma bloc VHDL 0](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781600)

[Figure 21 - GraphBit V1.0 : [1] - Alimentation 0](#_Toc10781601)

[Figure 22 - GraphBit V1.0 : [2] - Alimentation AP3012 0](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781602)

[Figure 23 - GraphBit V1.0 : [3] : Alimentation Tension Divider 1](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781603)

[Figure 24 - GraphBit V1.0 : [4] - Alimentation FPGA 2](#_Toc10781604)

[Figure 25 - GraphBit V1.0 : [5] - FPGA 3](#_Toc10781605)

[Figure 26 - GraphBit V1.0 : [6] - Clock 0](#_Toc10781606)

[Figure 27 - GraphBit V1.0 : [7] - EEPROM 0](#_Toc10781607)

[Figure 28 - GraphBit V1.0 : [8] - USB 0](#_Toc10781608)

[Figure 29 - GraphBit V1.0 : [9] - I2C 1](#_Toc10781609)

[Figure 30 - GraphBit V1.0 : [10] - Port Parallèle 2](#_Toc10781610)

[Figure 31 - GraphBit V1.0 : [11] - Écran 3](#_Toc10781611)

[Figure 32 - GraphBit V1.0 : [12] - SPI 4](#_Toc10781612)

[Figure 33 - GraphBit V1.0 : TOP 5](#_Toc10781613)

[Figure 34 - GraphBit : BOT 6](#_Toc10781614)

[Figure 35 - GraphBit : POWER 7](#_Toc10781615)

[Figure 36 - GraphBit V1.0 : SIGNAL 8](#_Toc10781616)

[Figure 37 - GraphBit V1.0 : SOLDERMASK TOP 9](#_Toc10781617)

[Figure 38 - GraphBit V1.0 : SOLDERMASK BOT 10](#_Toc10781618)

[Figure 39 - GraphBit V1.0 : SILKSCREEN TOP 11](#_Toc10781619)

[Figure 40 - GraphBit V1.0 : SILKSCREEN BOT 12](#_Toc10781620)

[Figure 41 - GraphBit V1.1 Change #1 - R2-2 13](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781621)

[Figure 42 - GraphBit V1.1 Change #2 - C2-2 13](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781622)

[Figure 43 - GraphBit V1.1 Change #3 - R3-0 14](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781623)

[Figure 44 - GraphBit V1.1 Change #4 - Condensateurs 0402 14](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781624)

[Figure 45 - GraphBit V1.1 Change #5 - Signal M2 14](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781625)

[Figure 46 - GraphBit V1.1 Change #6 - Signal V0 15](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781626)

[Figure 47 - GraphBit V1.1 Change #7 - Signal CE\_PROM 16](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781627)

[Figure 48 - GraphBit V1.1 : [8] - USB & UART 16](#_Toc10781628)

[Figure 49 - GraphBit V1.1 Change #9 - Connecteur à l'envers 17](#_Toc10781629)

[Figure 50 - Appels de fonction d'une écriture en mode texte 2](file:///E:\GitHub\GraphBitV1\Doc\GraphBit.docx#_Toc10781630)

[Figure 51 - Guidelines de création d'une specific deviation (MISRA-C:2004) 3](#_Toc10781631)

# Description du projet

Le projet GraphBit, réalisé par Charles Lévesque-Matte et Pascal-Emmanuel Lachance, est un contrôleur d’écran LCD ayant à son centre un FPGA. Son but est de pouvoir afficher sur un écran LCD monochrome 240x160, autant en mode graphique qu’en mode texte, des informations qui lui sont envoyées par des kits de développement ou un PC.

Notre contrôleur d’écran doit donc pouvoir communiquer avec le plus de périphériques possibles, ou plutôt, être lui-même périphérique au plus d’appareils possibles. C’est pour cela qu’on doit pouvoir l’interfacer à l’aide d’un grand nombre de protocoles de communication, c’est-à-dire :

* Parallèle
* USB
* UART
* I²C
* SPI

## Défi

Nous avons choisi ce projet en grande partie parce qu’il s’agissait d’un défi que nous lançait notre coach, Daniel Côté. Le défi de faire un contrôleur d’écran avait été donné à de nombreuses équipes dans le passé (2009, 2011, 2012 et 2013), et nous nous sentions prêts à reprendre le flambeau

## Kit étudiant

Une autre raison pour laquelle nous avons choisi de faire un contrôleur d’écran est la possibilité que ce dernier devienne un kit pour les étudiants dans les sessions à venir. C’est une des raisons pour lesquelles nous voulions lui offrir tant de protocoles de communication, afin de pouvoir être utilisé dans de nombreux contextes sans problèmes et être très versatile. Créer un kit étudiant était une grande motivation car nous léguions quelque chose au programme et à la postérité de TSO. Cependant, elle amenait avec elle de nombreux autres défis, au-delà de la versatilité du projet, il fallait également penser au coût de cette carte, et surtout la rendre à l’épreuve des étudiants, capables des pires erreurs.

### Protections

Afin de nous protéger des étudiants, au-delà de créer un boîtier pour contenir le projet (pour le protéger de courts-circuits qui peuvent être entraînés par de mauvaises manipulations et des fils dénudés), nous avons également isolés le FPGA de toutes ses entrées en employant des transistors qui ont deux buts : Transiter d’un niveau logique 5V au 3.3V nécessaire pour les entrées du FPGA, et également d’agir comme une ligne de défense entre les connecteurs et le FPGA dans le cas de courts-circuits, de décharge de statique, ou autre malheur. Des résistances de terminaisons sont également ajoutées au connecteur parallèle afin d’empêcher les entrées de ce dernier d’être flottantes lorsque rien n’y est connecté. Finalement, une quantité aberrante de condensateurs sont présentes sur le PCB afin d’assurer une tension incroyablement stable et robuste au travers des divers niveaux de tensions.

### Conformité MISRA-C : 2004

Un défi supplémentaire que nous nous sommes lancés était de rendre notre code en C complètement conforme aux normes de programmations MISRA-C :2004. Rendre les librairie C conformes aux normes MISRA-C :2004 veut également dire que ces dernières sont encore plus robustes que notre matériel, et à l’épreuve du pire que les étudiants peuvent faire. Un avantage d’utiliser une telle norme est que notre code C a presque entièrement marché du tout premier coup, alors qu’il s’agissait d’un des plus gros codes que l’on aie écrit.



Figure – PCB GraphBit V1.0 montrant fièrement son FPGA

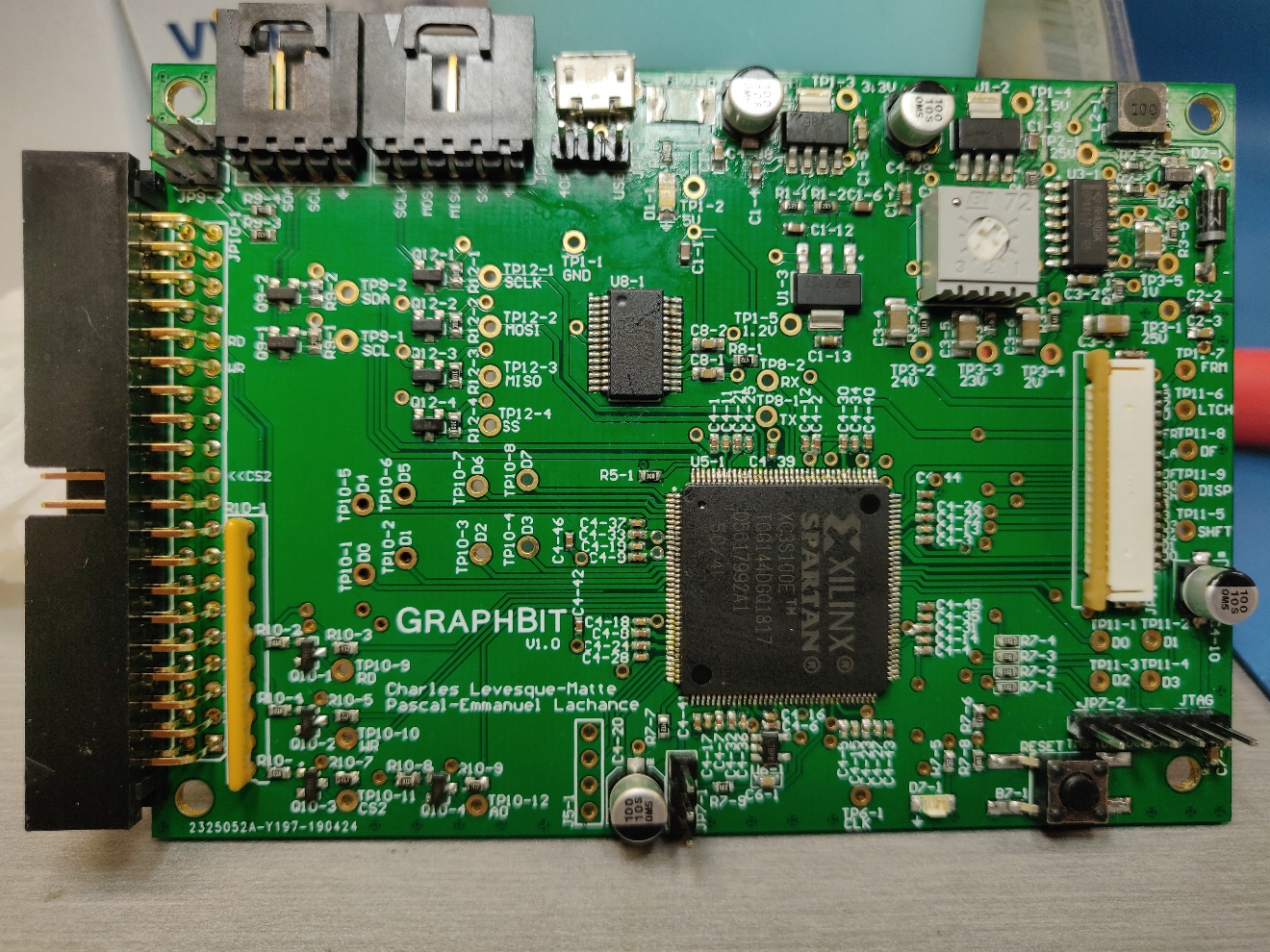
# Description fonctionnelle du projet

Figure - Disposition des éléments principaux sur le PCB GraphBit V1.0

I2C

SPI

USB



Parallèle

Écran

FPGA

## FPGA

Au cœur du projet se trouve un FPGA Xilinx Spartan3E ([XC3S100E](https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf)), le même modèle qui se retrouve sur le kit de développement Basys. Ce dernier traite toutes les commandes entrantes et affiche constamment à l’écran, à une fréquence de rafraîchissement de 70Hz. L’emploi d’un FPGA permet d’effectuer toutes les acquisitions de données et l’affichage simultanément, ce qui serait impossible sur un microcontrôleur standard, surtout à hautes fréquences de communications (telles que le 921600bauds qui est la vitesse maximale qu’un ordinateur peut fournir sur le port série, que le FPGA traite facilement, sans broncher).

### JTAG

Pour téléverser un programme dans le FPGA et la mémoire EEPROM ([XCF02S](https://www.xilinx.com/support/documentation/data_sheets/ds123.pdf)) l’accompagnant, un connecteur JTAG se trouve sur le PCB. Pour s’y brancher, il faut employer un câble de programmation JTAG de Digilent, disponible au magasin de TGE. Les pilotes de ce câble ne sont malheureusement pas à jour, et pour utiliser le logiciel iMPACT de Xilinx avec le câble, il faut un ordinateur roulant Windows 32-bits.

## Écran

La raison d’être de ce projet est l’écran LCD qui trône en son centre.

Il s’agit d’un écran monochrome 240x160, sans contrôleur d’écran, et avec peu de documentation. Le Cégep Limoilou en reçu un don dans le coin de 2008, sous forme d’une boîte absolument pleine de ces petits écrans LCD, et ils ont depuis amassés la poussière.

Mais plus maintenant! En les connectant à notre projet, nous pouvons enfin donner vie à ces écrans capable de rafraîchir à 75Hz.

(1, 1) (240,1)

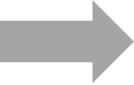
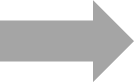
(1,160) (240,160)

Figure - Coordonnées des pixels aux extrémités de l'écran

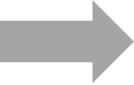
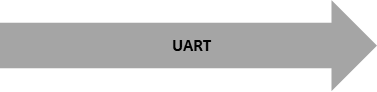
## Entrées

### USB

Figure - Schéma bloc USB



USB



Jumper

USB / UART

FPGA

FTDI

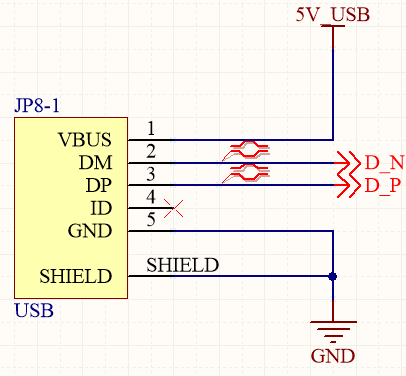
Le connecteur USB est l’un des deux connecteurs principaux du contrôleur d’écran, car il apporte non seulement l’alimentation 5V principale du PCB, mais il transporte également des signaux de données série qui sont décodées par un FTDI ([FTDI232RL](https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf)) vers du UART transmis vers le FPGA, qui est capable de décoder ces signaux pouvant passer de 9600bauds à 921600bauds, grâce à un module UART VHDL faisant de l’autobaud.

Figure - Connecteur USB

### Parallèle

Figure - Schéma bloc parallèle

Port Parallèle

Adresse & CS

Data

5V

5V -> 3.3V

(74LVC4245)

5V -> 3.3V

FPGA

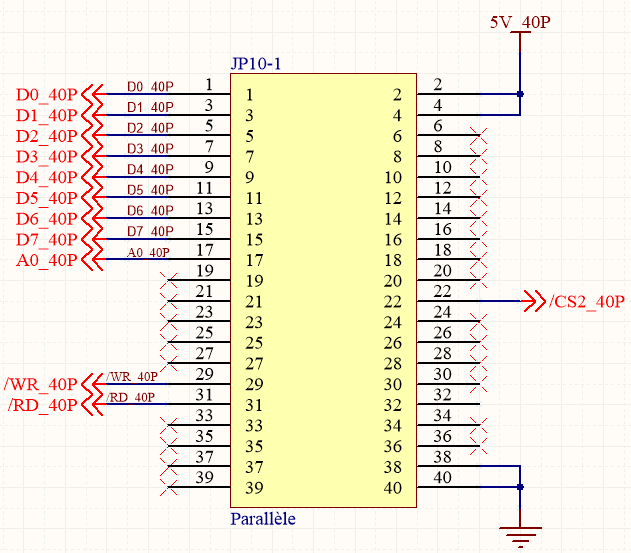
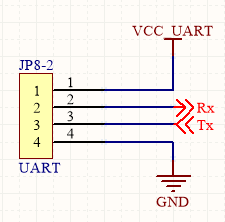
Le port parallèle est le deuxième connecteur principal du projet, apportant lui également une alimentation 5V, en plus de nombreux signaux de données, d’adresse et de chip select.

Figure - Connecteur Parallèle

Les signaux de données passent en premier dans une résistance de terminaison, les empêchant d’être flottantes, puis passent dans un level shifter 8-bits ([74LVC4245](http://www.ti.com/lit/ds/symlink/sn74lvc4245a.pdf)) avant d’entrer dans le FPGA.

Les signaux d’adresse, de chip select, READ et WRITE, eux, vont directement à des level shifters faits à base de simples transistors bipolaires ([MMBT3904](https://www.onsemi.com/pub/Collateral/MMBT3904WT1-D.PDF)), pour ensuite aller au FPGA.



### UART

En plus de supporter du UART via USB, la version 1.1 du contrôleur d’écran comprend également un simple connecteur UART, servant par exemple à connecter un microcontrôleur à l’écran via un simple lien série.

### I²C

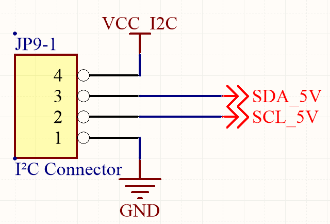
Le contrôleur d’écran agit également comme un esclave I²C, à l’adresse 0x52, et possède un connecteur I²C permettant un lien bidirectionnel avec un microcontrôleur externe. Les signaux I²C sont séparés du FPGA par un MOSFET ([NTR5103N](https://www.onsemi.com/pub/Collateral/NTR5103N-D.PDF)) agissant également comme level shifter.

Figure - Connecteur I2C

Figure - Connecteur UART

### SPI

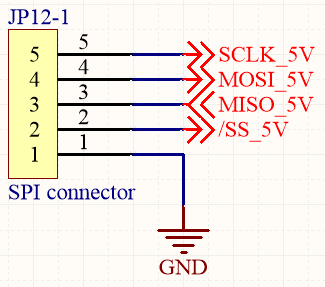
Le circuit SPI fonctionne exactement comme le circuit I²C, en étant un bus bidirectionnel entre un connecteur molex et le FPGA, séparé par un MOSFET du FPGA.

Figure - Connecteur SPI

### Trame de communication

Peu importe quel protocole est utilisé, les trames envoyées au contrôleur doivent êtres sous le même format.

En premier, on envoie la taille de la trame, sur 16 bits, donc en deux paquets.

Puis, on envoie un octet indiquant le mode dans lequel on souhaite mettre l’écran.

Si l’on est en mode graphique, on commence immédiatement à transmettre de données. Cependant, dans les autres modes, on envoie en premier la ligne, puis la colonne à laquelle on souhaite commencer à écrire.

Tous les autres octets consécutifs sont des données, jusqu’au dernier octet, qui est un simple checksum 8-bits qui n’est pas utilisé.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mode | Octet1 | Octet2 | Octet3 | Octet4 | Octet5 | Octet6 | Octet(n-1) | Octet n |
| **Graph** | Taille0 | Taille1 | Mode | Data0 | Data1 | Data2 | Data(n-1) | Checksum |
| **Texte** | Taille0 | Taille1 | Mode | Ligne | Colonne | Data0 | Data(n-1) | Checksum |
| **Pixel** | Taille0 | Taille1 | Mode | Ligne | Colonne | Data1 | Data(n-1) | Checksum |

## Modes

### https://scontent.fyhu1-1.fna.fbcdn.net/v/t1.15752-9/s2048x2048/61667264_357624584892437_8334581975750803456_n.jpg?_nc_cat=105&_nc_ht=scontent.fyhu1-1.fna&oh=79f2a0689235e77c9d0a9c10aadbaf53&oe=5D84EF4FMode Graphique

D’un contrôleur d’écran LCD, on attend au moins un mode graphique permettant l’affichage d’images et même d’animations fluides. En mode graphique, on peut donc envoyer un tableau via n’importe quel protocole de communication supporté, dans lequel chaque octet de data représente 8 pixels consécutifs à partir de l’origine. Sur l’écran 240x160, cela équivaut donc à 4800 octets.

Figure - Mode graphique

### Mode Texte

Figure - Mode graphique et texte mélangés

Digne d’une carte graphique plus que d’un contrôleur d’écran, le projet GraphBit fait également de l’accélération matérielle, car une police d’écriture 8x8 est chargée dans la mémoire du FPGA, et ce dernier peut recevoir de simples codes ASCII afin d’écrire du texte à l’écran. Cela permet d’éliminer une charge de travail considérable du microcontrôleur, ainsi que de raccourcir les trames, permettant une communication encore plus rapide.

## Librairie C

### LCD

La bibliothèque de code LCD permet de paramétrer divers protocoles de communication, et d’envoyer des images, animations et textes à l’écran.

#### Exemple de code :

#include <string.h>

#include "LCD.h"

void main(void)

{

static const char Message[] = "Bien le bonjour!";

// Initialize parallel bus with no timer

LCD\_Init\_Parallel(NULL);

// Write "Bien le bonjour!" on the second row and first column of the LCD

LCD\_WriteText(Message, strlen(Message), 1, 0);

}

### Parallel

Sur la carte STM, il n’y avait pas de bus parallèle parmi ses périphériques, ce qui rend la communication parallèle légèrement problématique. J’ai donc également créé une librairie de code pour créer un port parallèle artificiel sur la STM.

#### Exemple de code :

#include "main.h"

#include "stm32f4xx\_hal.h”

#include "Parallel.h"

void main(void)

{

// HAL configuration

// GPIO pins will be configured by the parallel library

HAL\_Init();

SystemClock\_Config();

// Configure 8-bits data pins : PE8, PE9, PE10, PE11, PE12, PE13, PE14, PE15

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);

// Configure 5-bits adress pins : PD8, PD9, PD10, PD11, PB1

Parallel\_InitAdress(5, GPIOD, GPIO\_PIN\_8, GPIOD, GPIO\_PIN\_9, GPIOD, GPIO\_PIN\_10, GPIOD, GPIO\_PIN\_11,

GPIOB, GPIO\_PIN\_1);

// Configure 4 CS pins : PB11, PB12, PB13, PB14

Parallel\_InitChipSelect(4, GPIOB, GPIO\_PIN\_11, GPIOB, GPIO\_PIN\_12, GPIOB, GPIO\_PIN\_13, GPIOB,

GPIO\_PIN\_14);

// Configure PB0 as a read pin, with no write pins enabled

Parallel\_InitReadWrite(GPIOB, GPIO\_PIN\_0, NULL, 0);

// Initialize parallel bus and configure GPIO pins

Parallel\_Init();

// Write 0x55 at adress 0x01 of chip select 0x02

Parallel\_Write(0x55, 0x01, 0x02);

}

## Logiciel C#

## Site Web

# Schéma bloc général

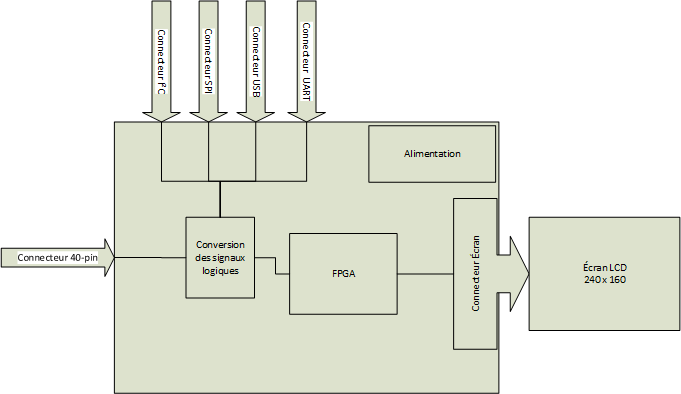


Figure - Schéma bloc général

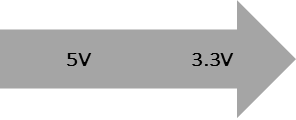
# Schéma blocs électriques

## Schéma général

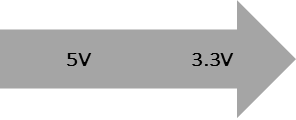
## Entrées

### I2C & SPI

Figure 14 - Schéma bloc I2C & SPI



I²C



SPI

FPGA

(XC3S100E)

### Parallèle

Port Parallèle

Adresse & CS

Data

5V

5V -> 3.3V

(74LVC4245)

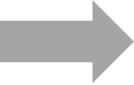
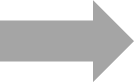
5V -> 3.3V

FPGA

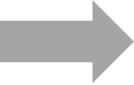
Figure - Schéma bloc parallèle

### USB & UART

UART



USB



Jumper

USB / UART

FPGA

FTDI

Figure - Schéma bloc USB

## Sorties

Figure - Schéma bloc connecteur écran

## Alimentation

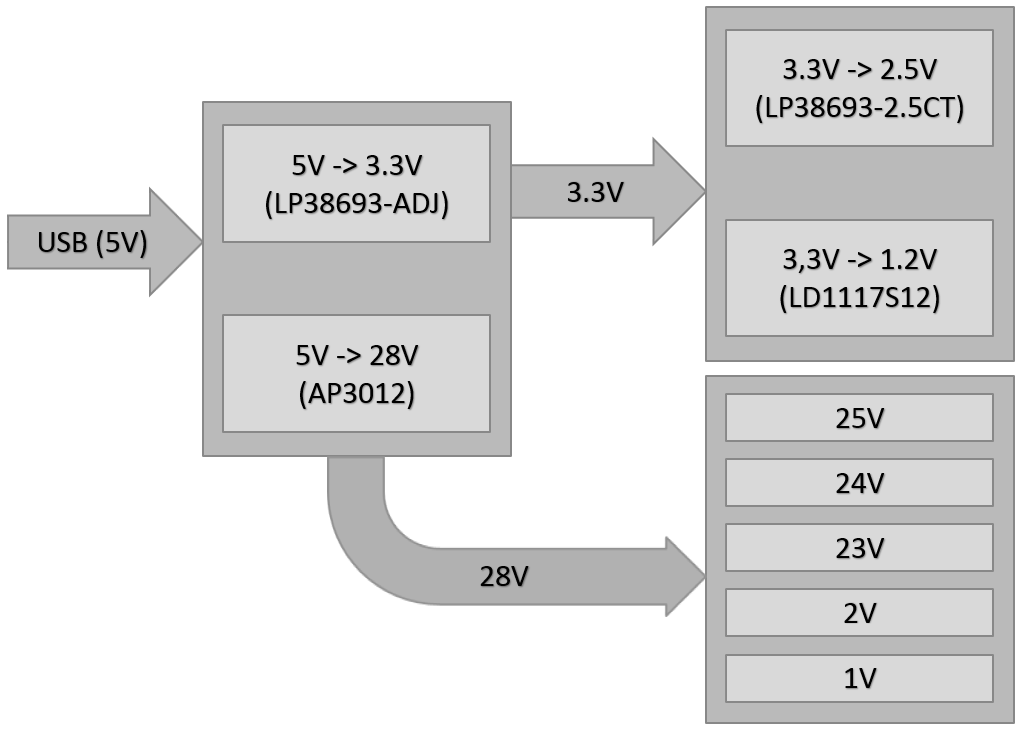


Figure - Schéma bloc alimentation

# Schéma blocs VHDL

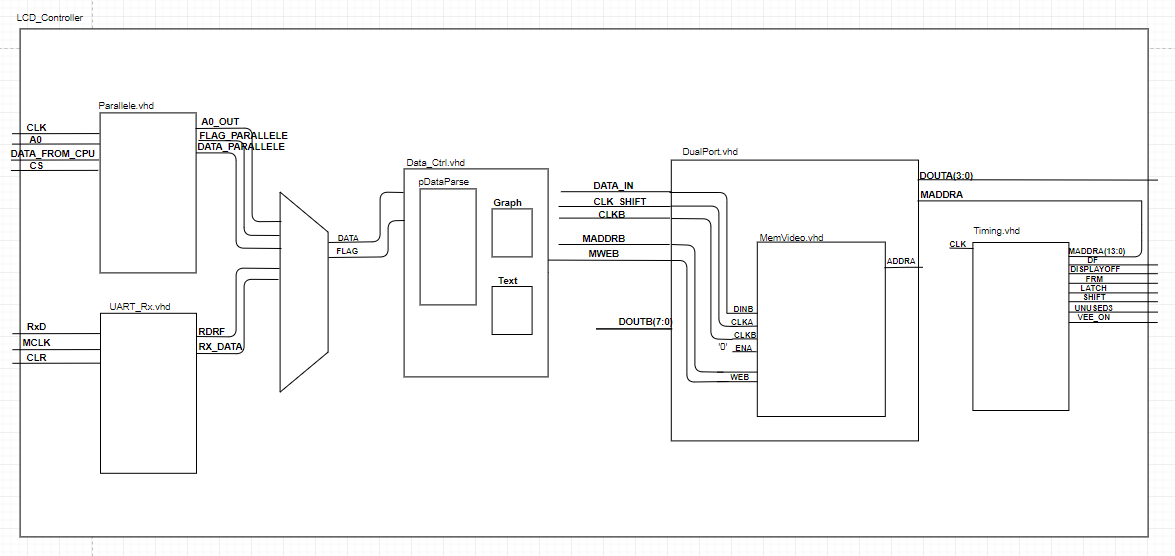
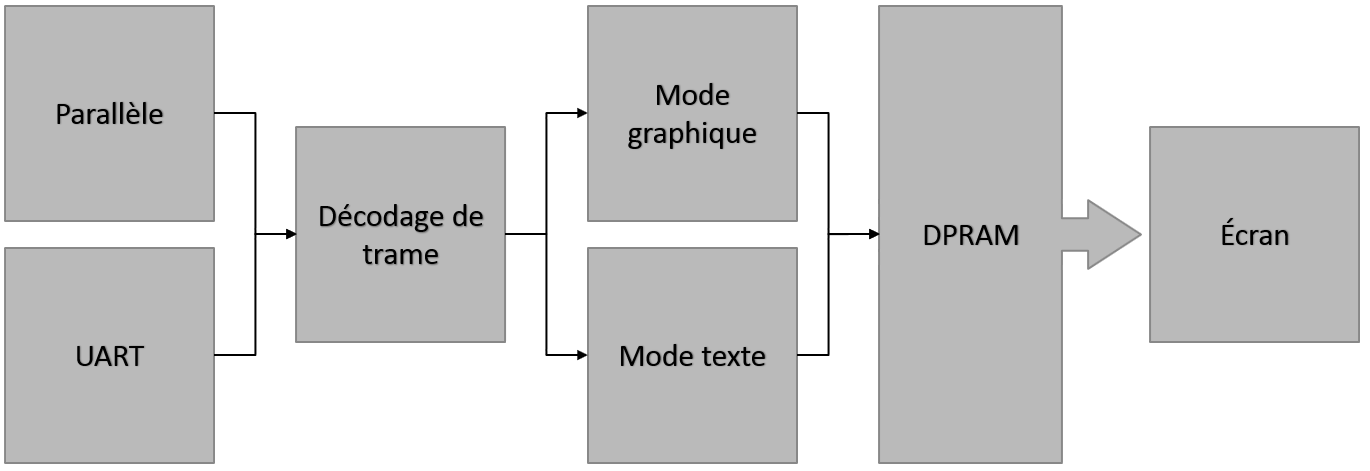


Figure - Schéma bloc VHDL simplifié

Figure - Schéma bloc VHDL

# Schémas électriques V1.0

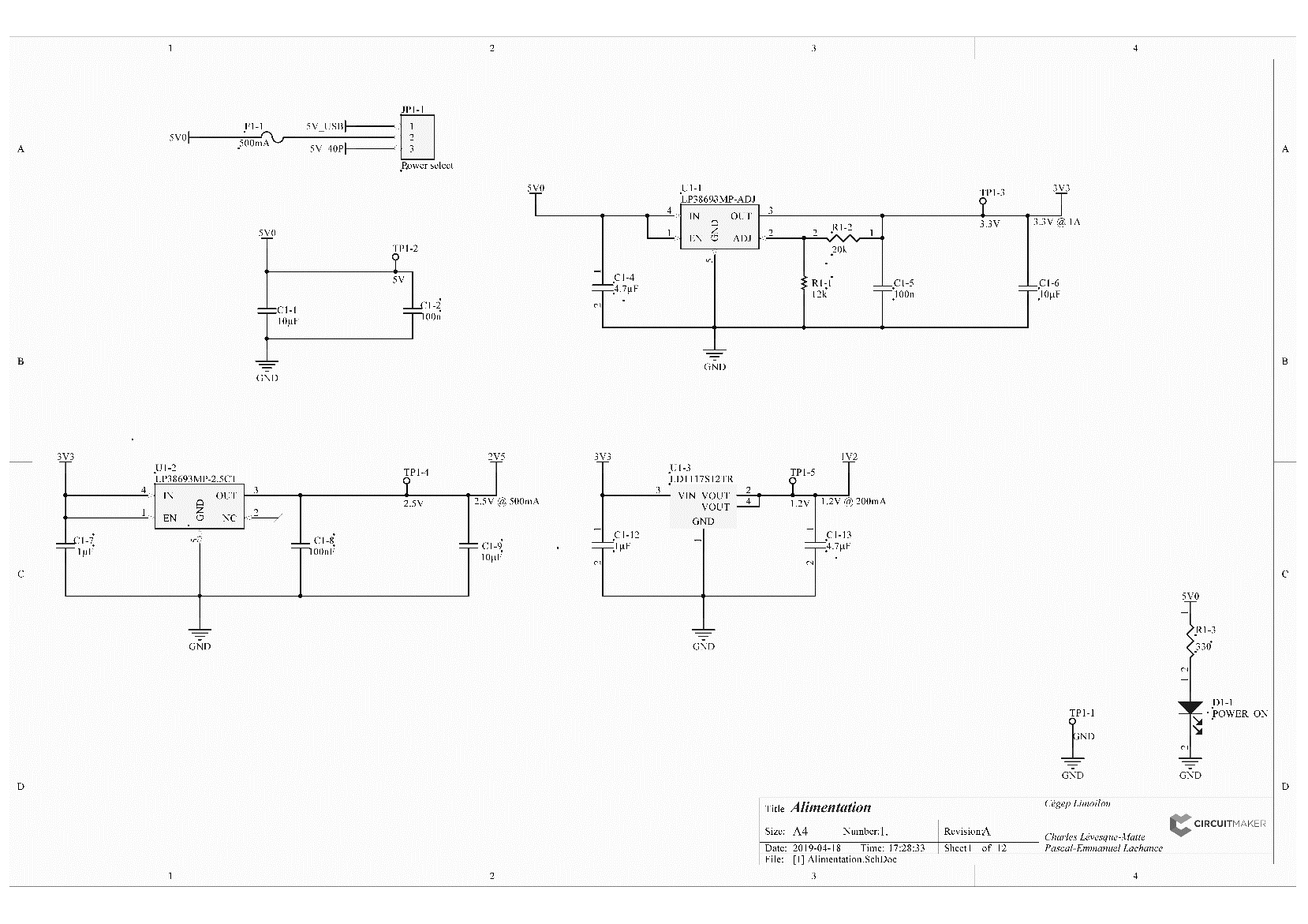


Figure - GraphBit V1.0 : [1] - Alimentation

****

Figure - GraphBit V1.0 : [2] - Alimentation [AP3012](https://www.diodes.com/assets/Datasheets/AP3012.pdf)

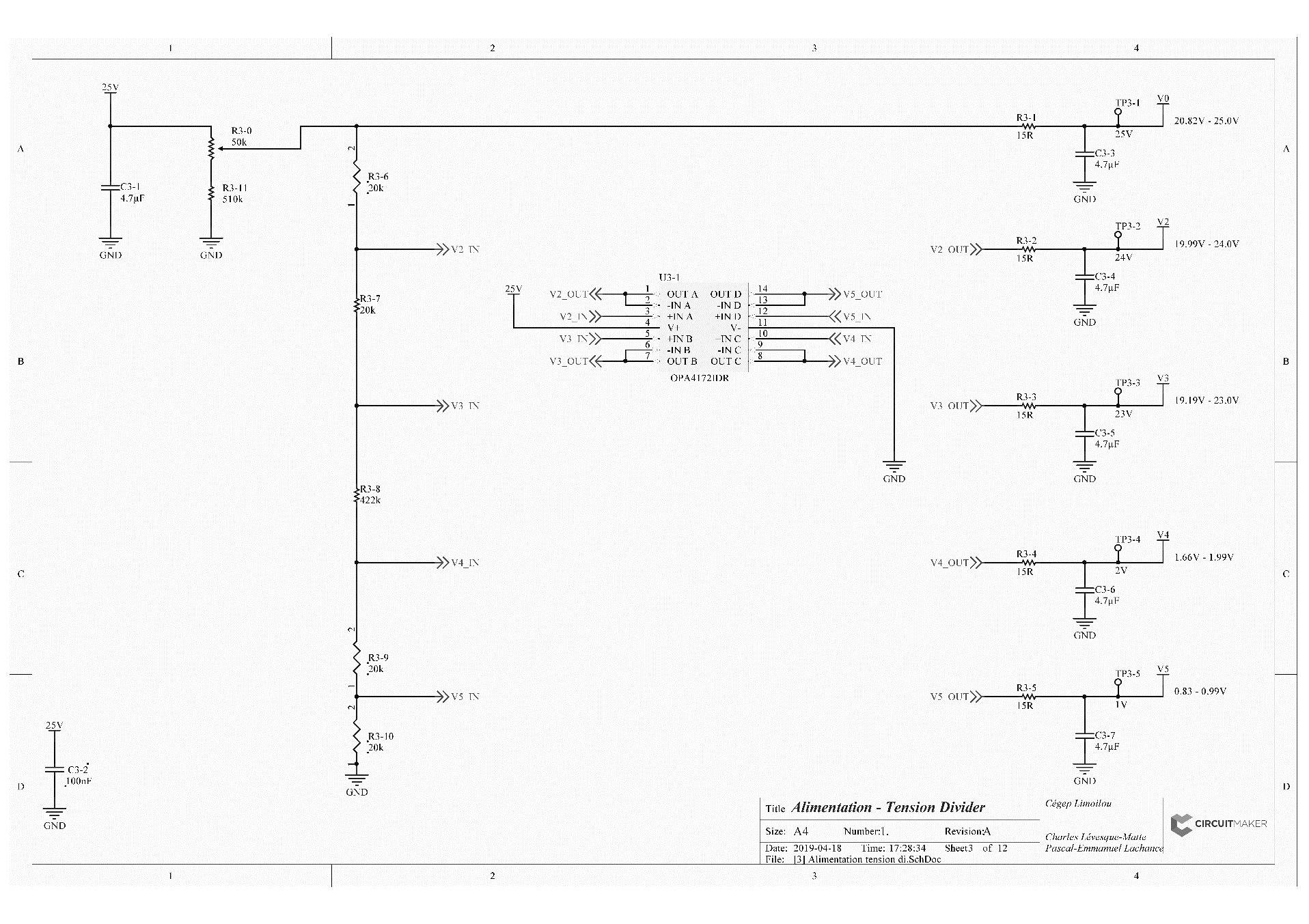
****

Figure - GraphBit V1.0 : [3] : Alimentation Tension Divider

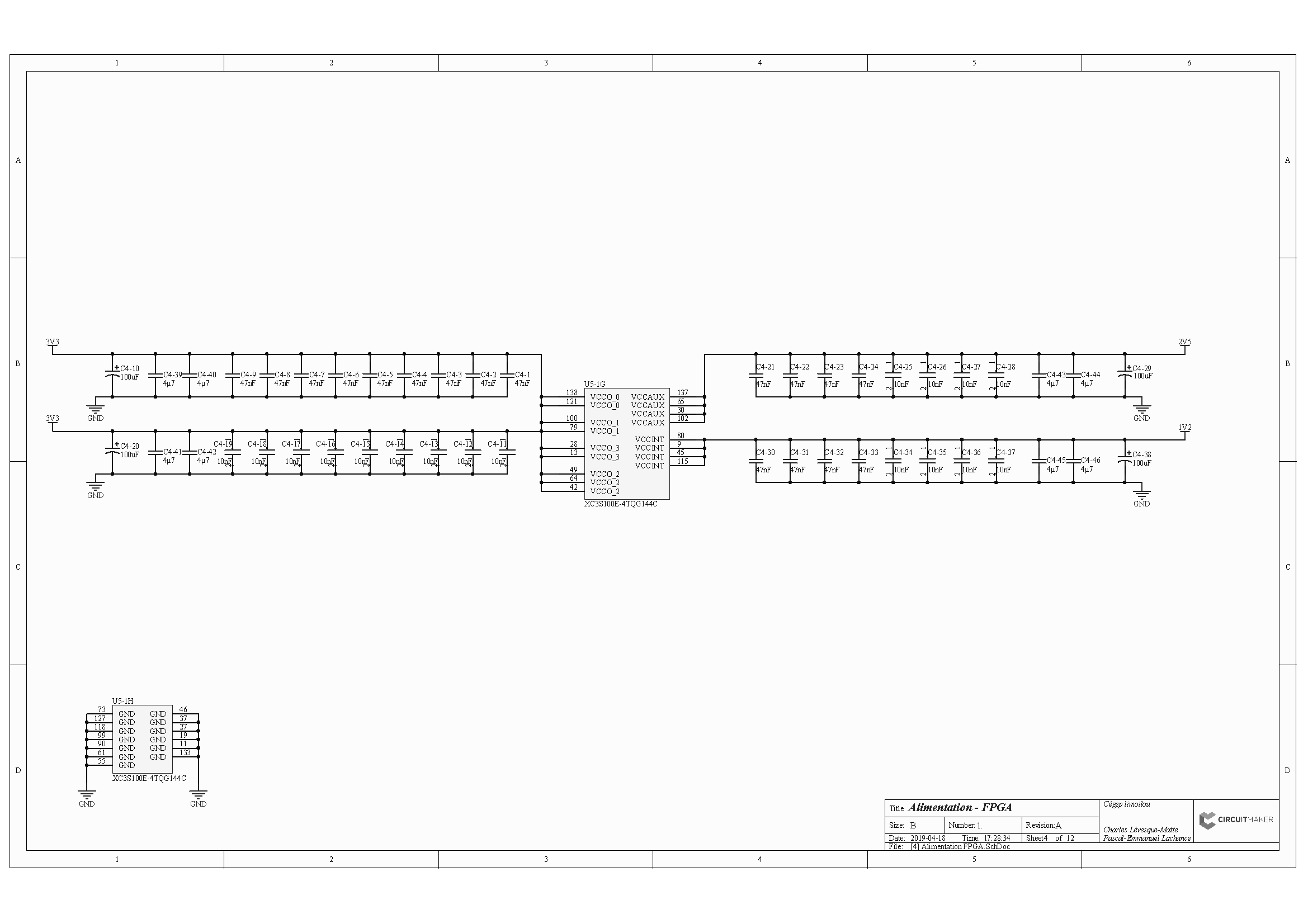
****

Figure - GraphBit V1.0 : [4] - Alimentation FPGA

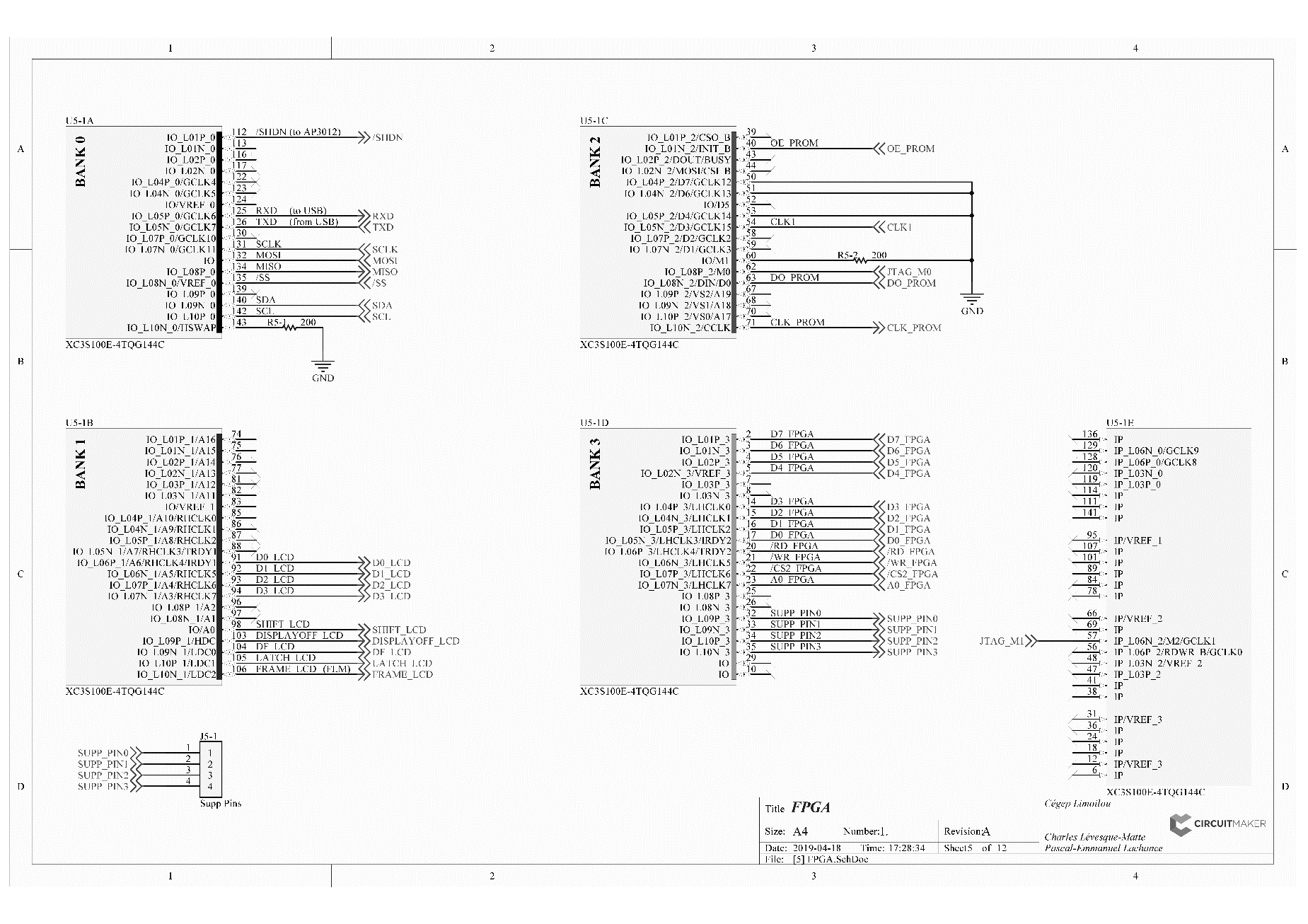
****

Figure - GraphBit V1.0 : [5] - FPGA

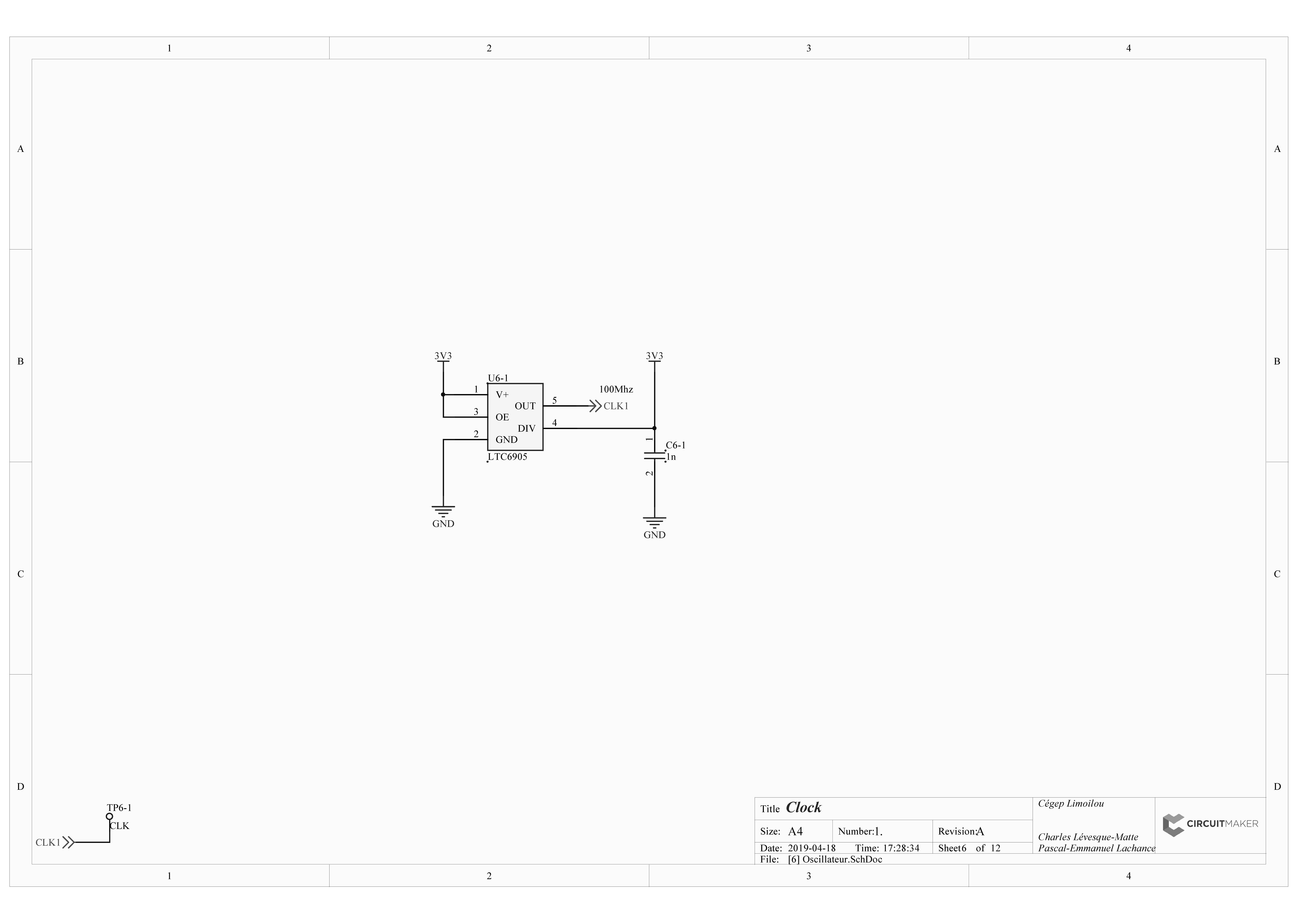
****

Figure - GraphBit V1.0 : [6] - Clock

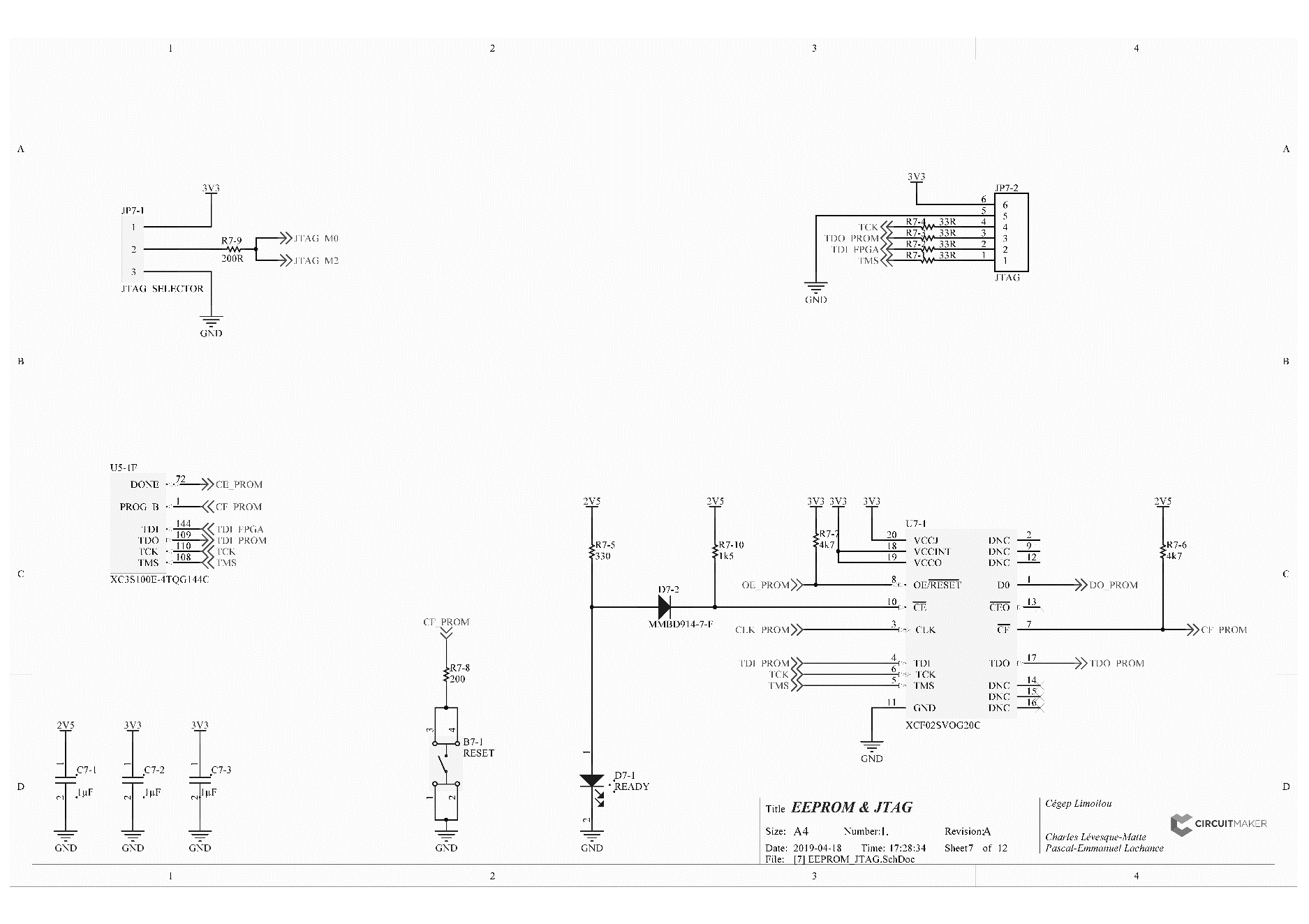
****

Figure - GraphBit V1.0 : [7] - EEPROM

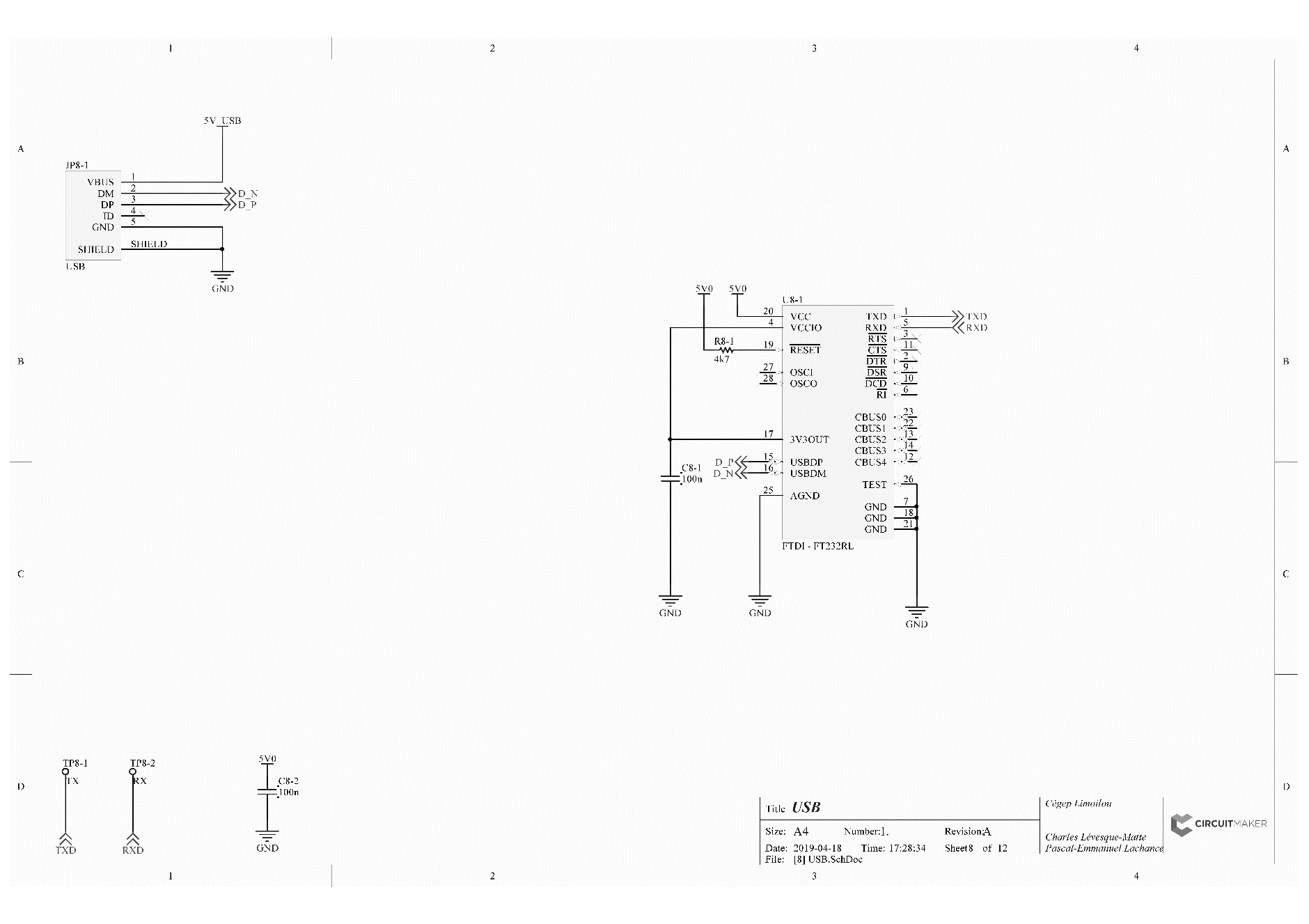
****

Figure - GraphBit V1.0 : [8] - USB

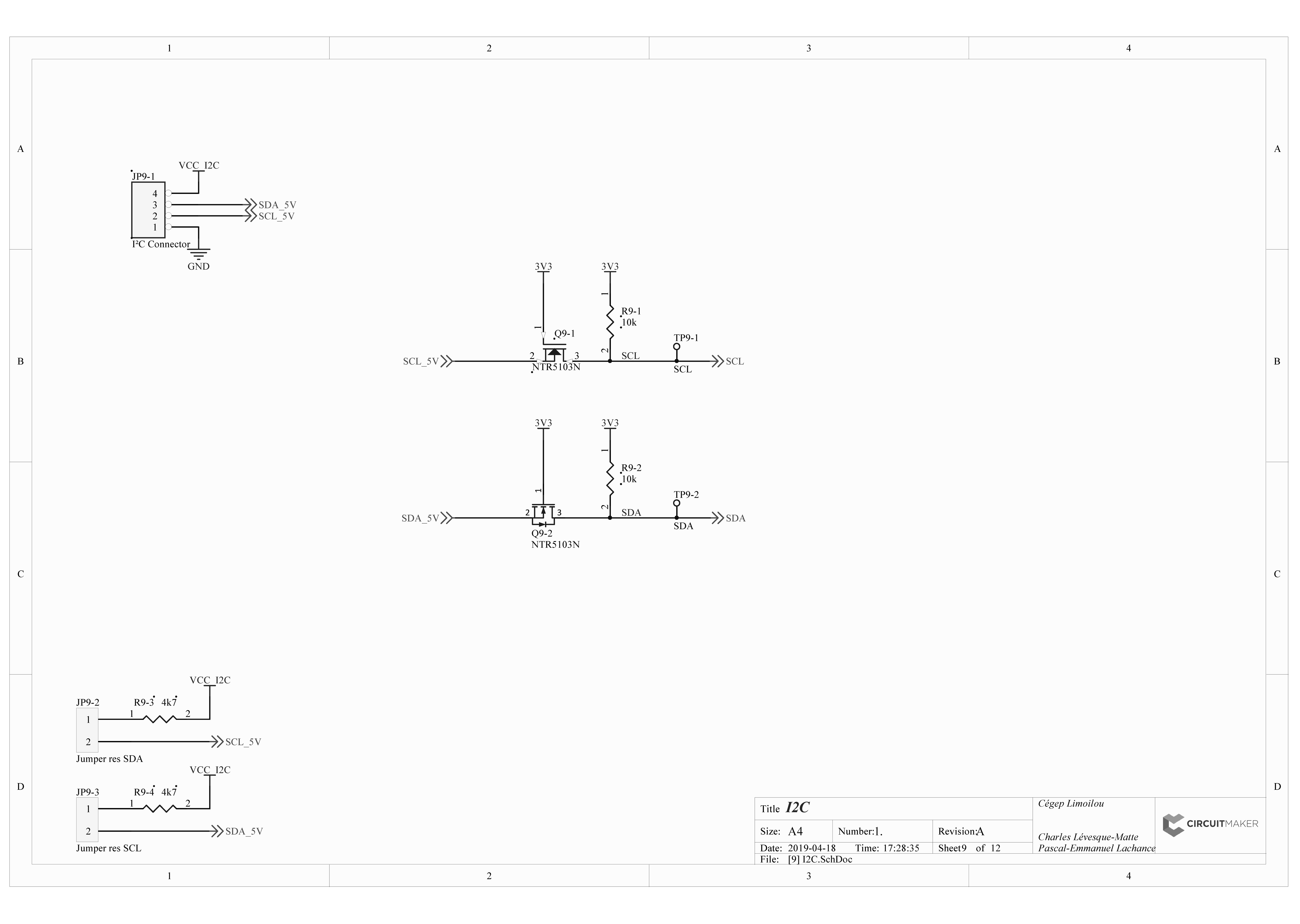
****

Figure - GraphBit V1.0 : [9] - I2C

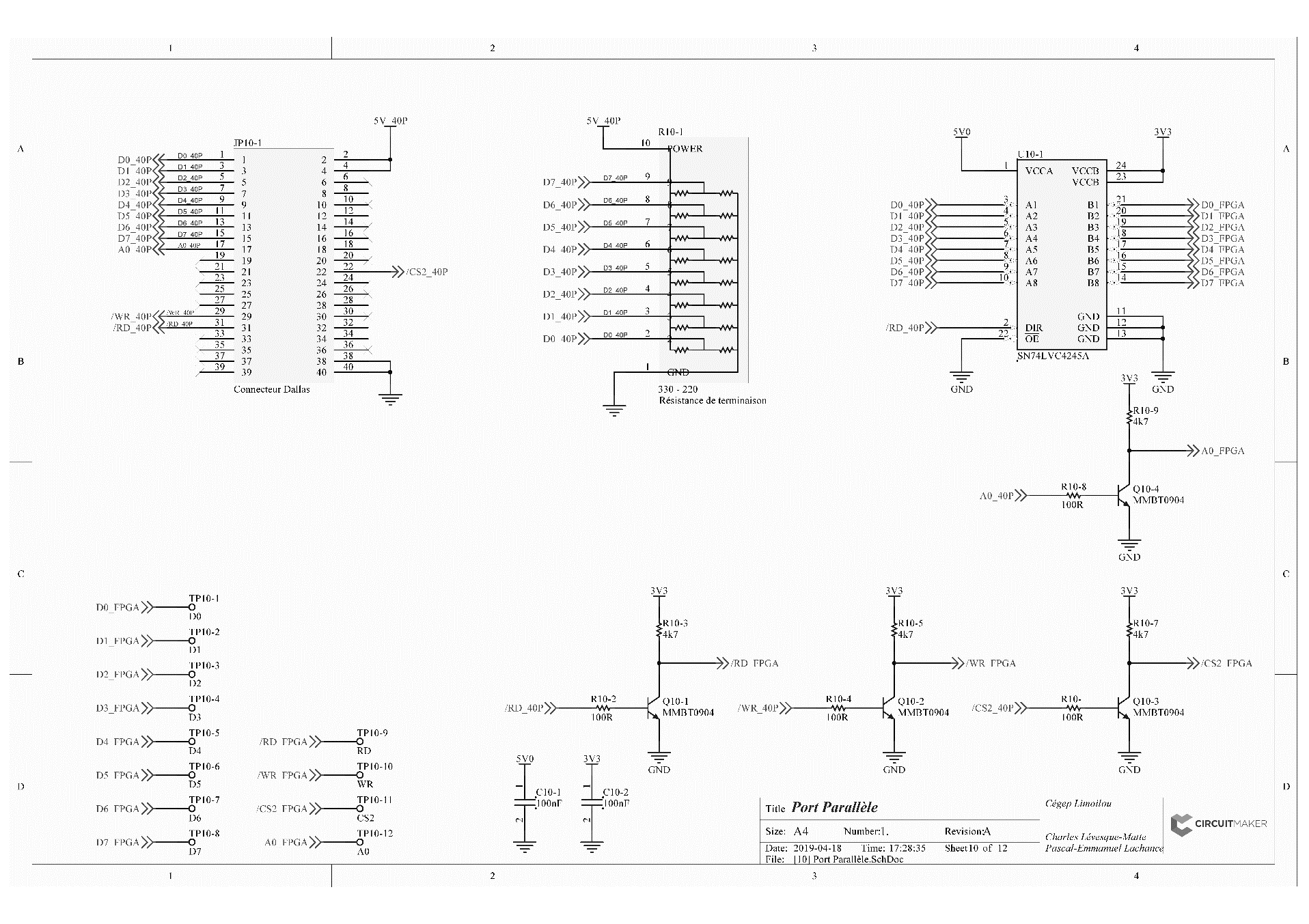
****

Figure - GraphBit V1.0 : [10] - Port Parallèle

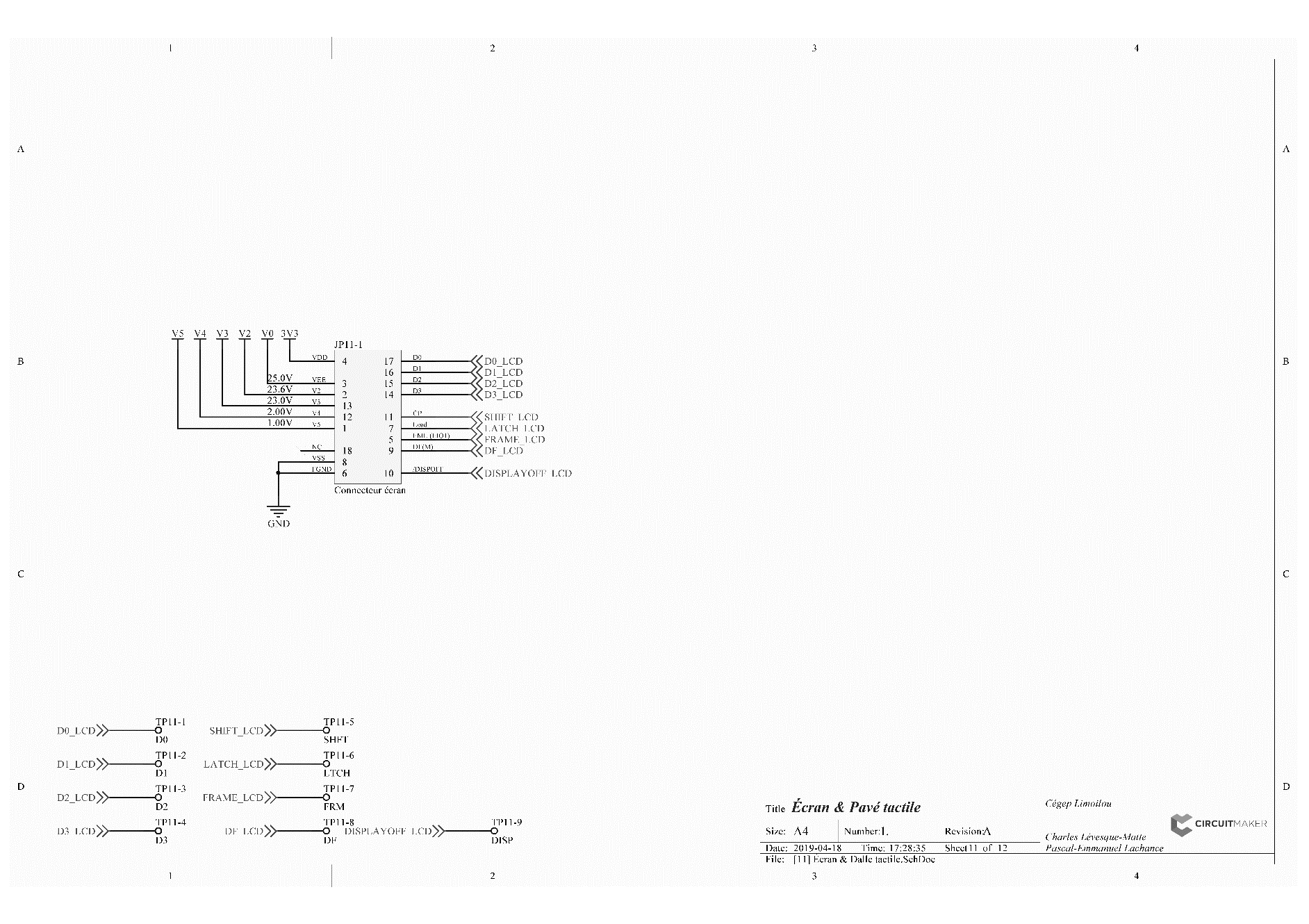
****

Figure - GraphBit V1.0 : [11] - Écran

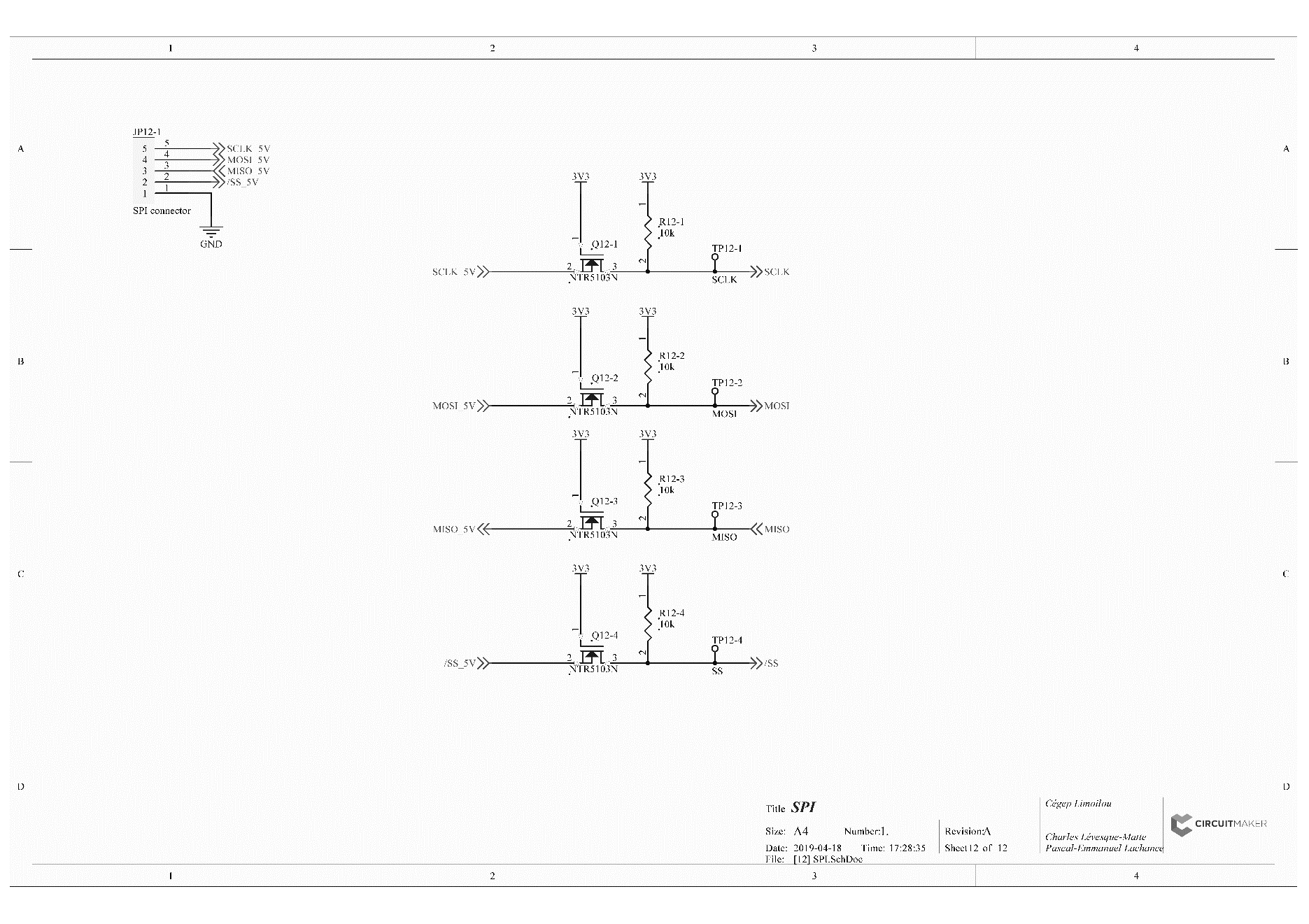
****

Figure - GraphBit V1.0 : [12] - SPI

# PCB V1.0



Figure - GraphBit V1.0 : TOP

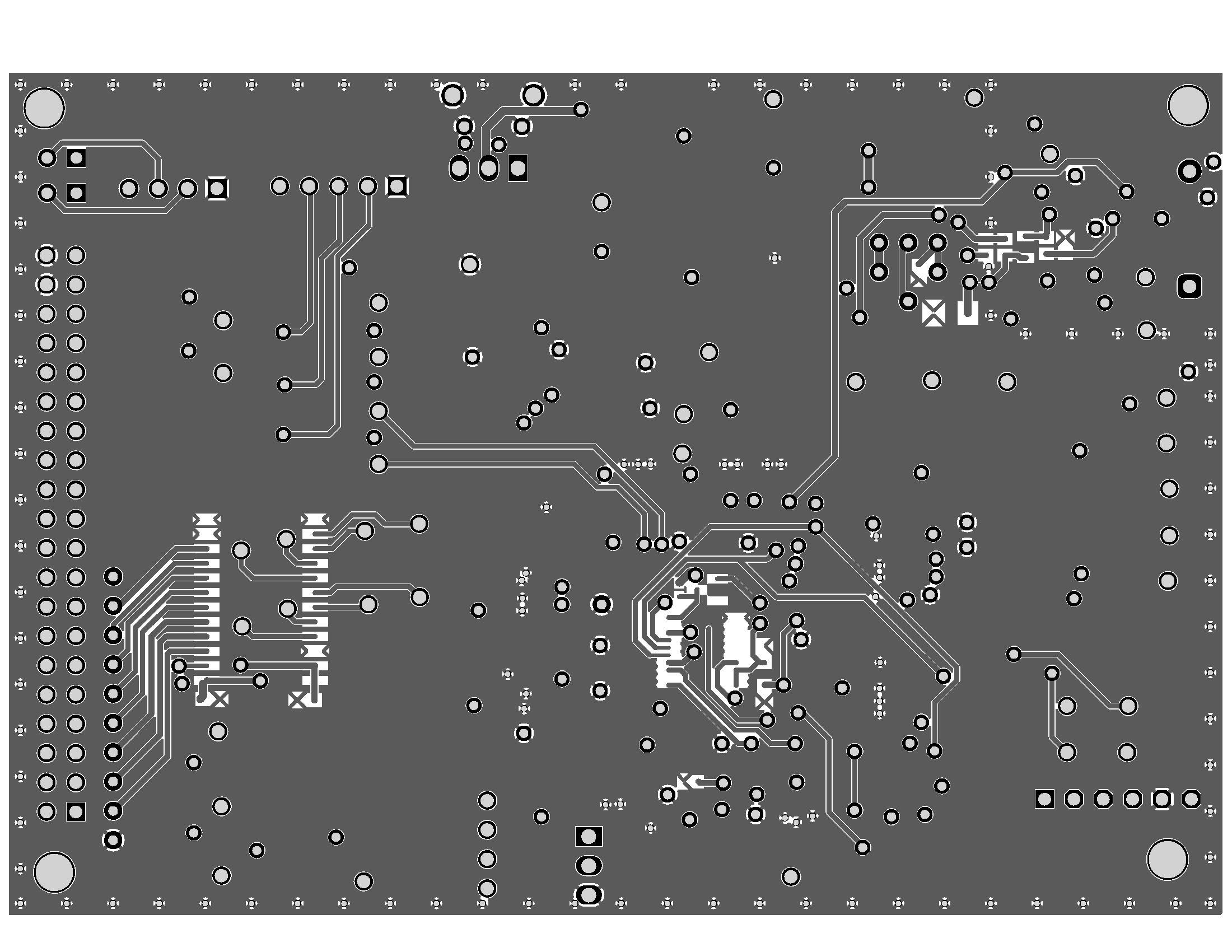


Figure - GraphBit : BOT

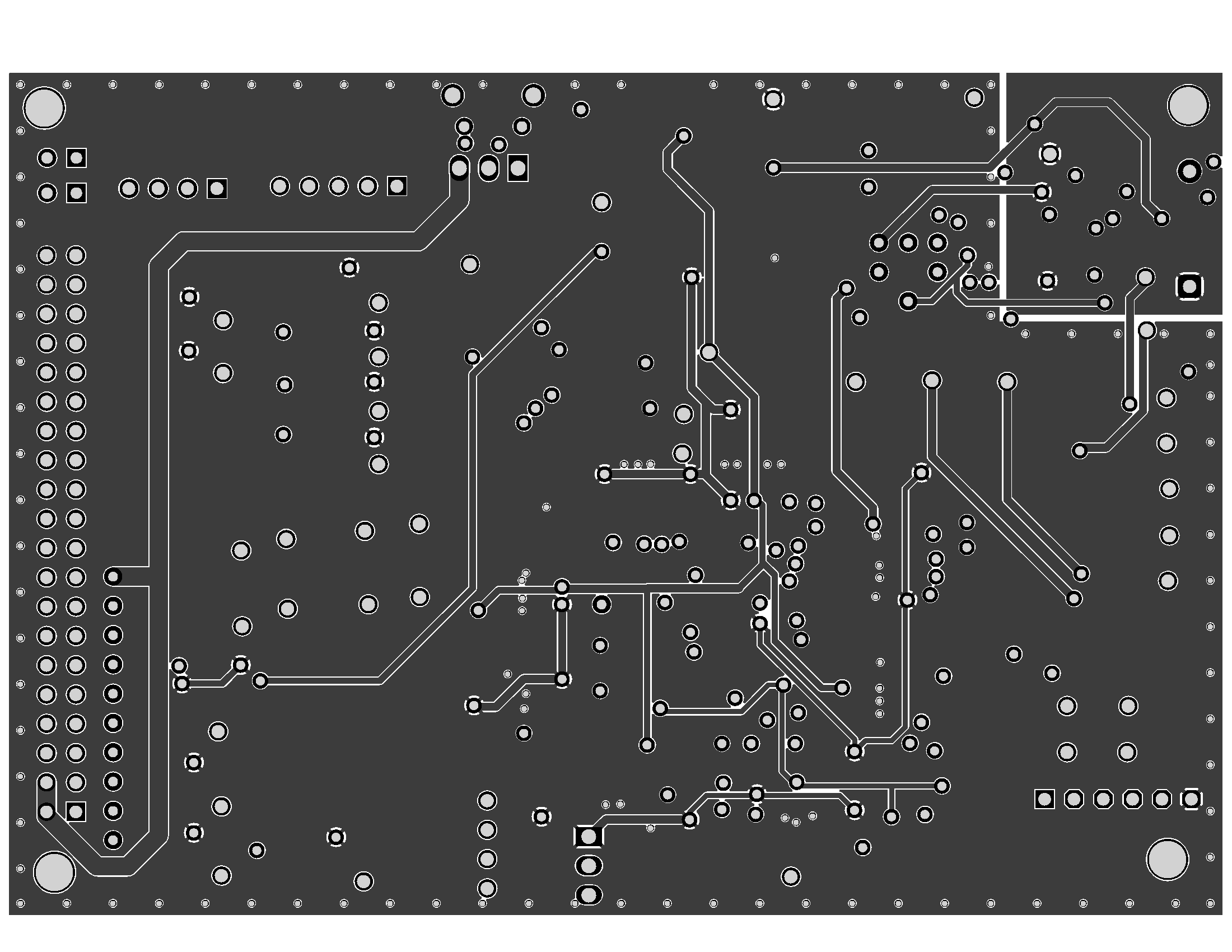


Figure - GraphBit : POWER

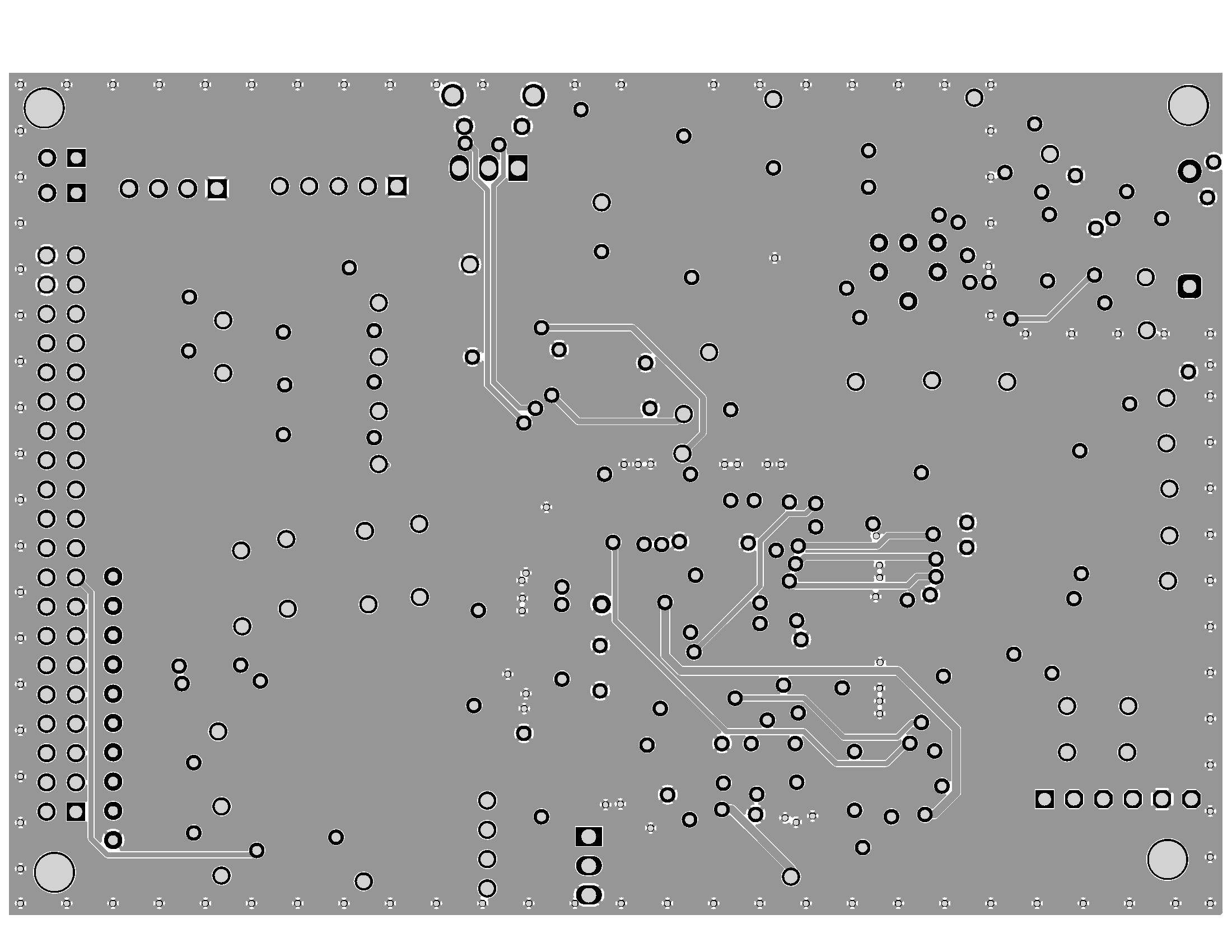


Figure - GraphBit V1.0 : SIGNAL

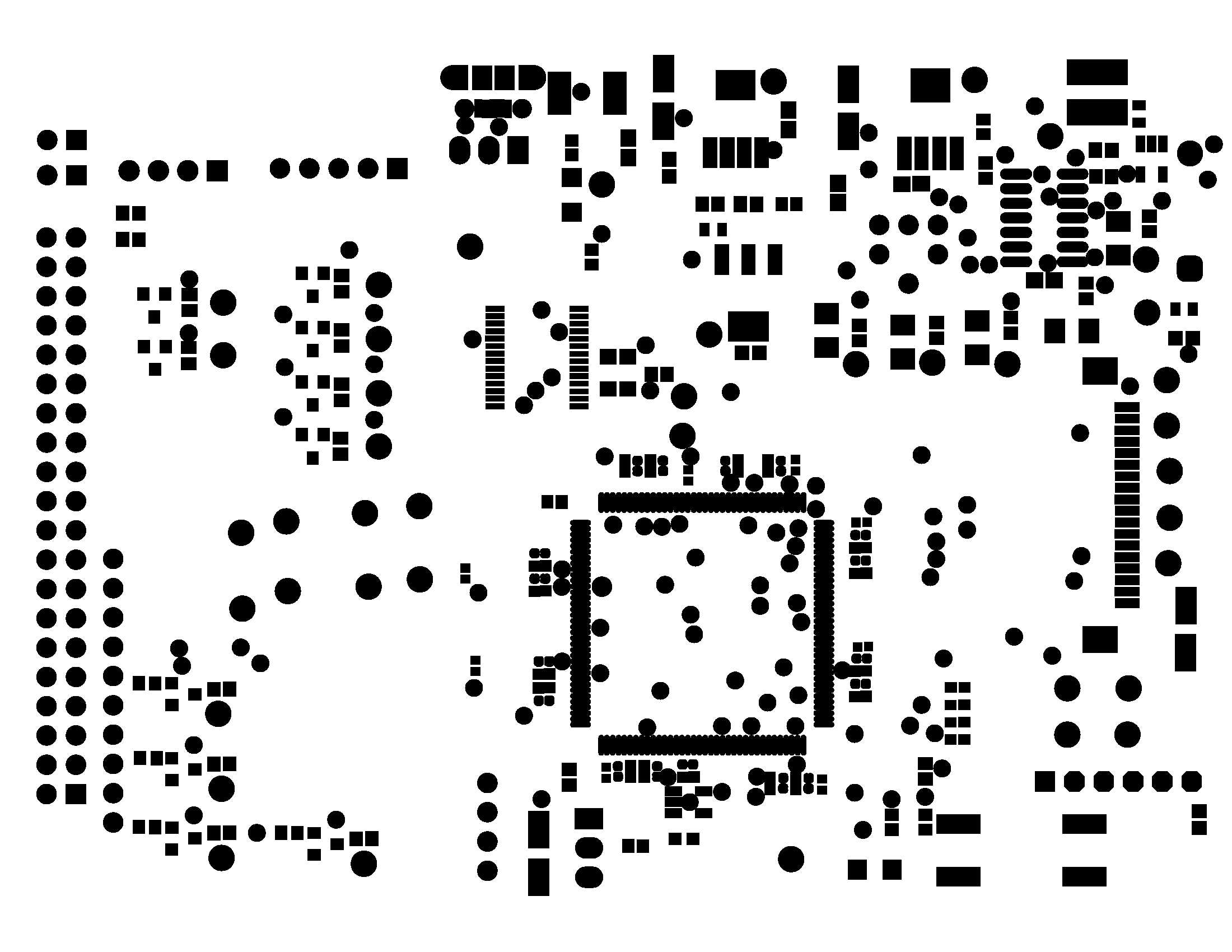


Figure 37 - GraphBit V1.0 : SOLDERMASK TOP

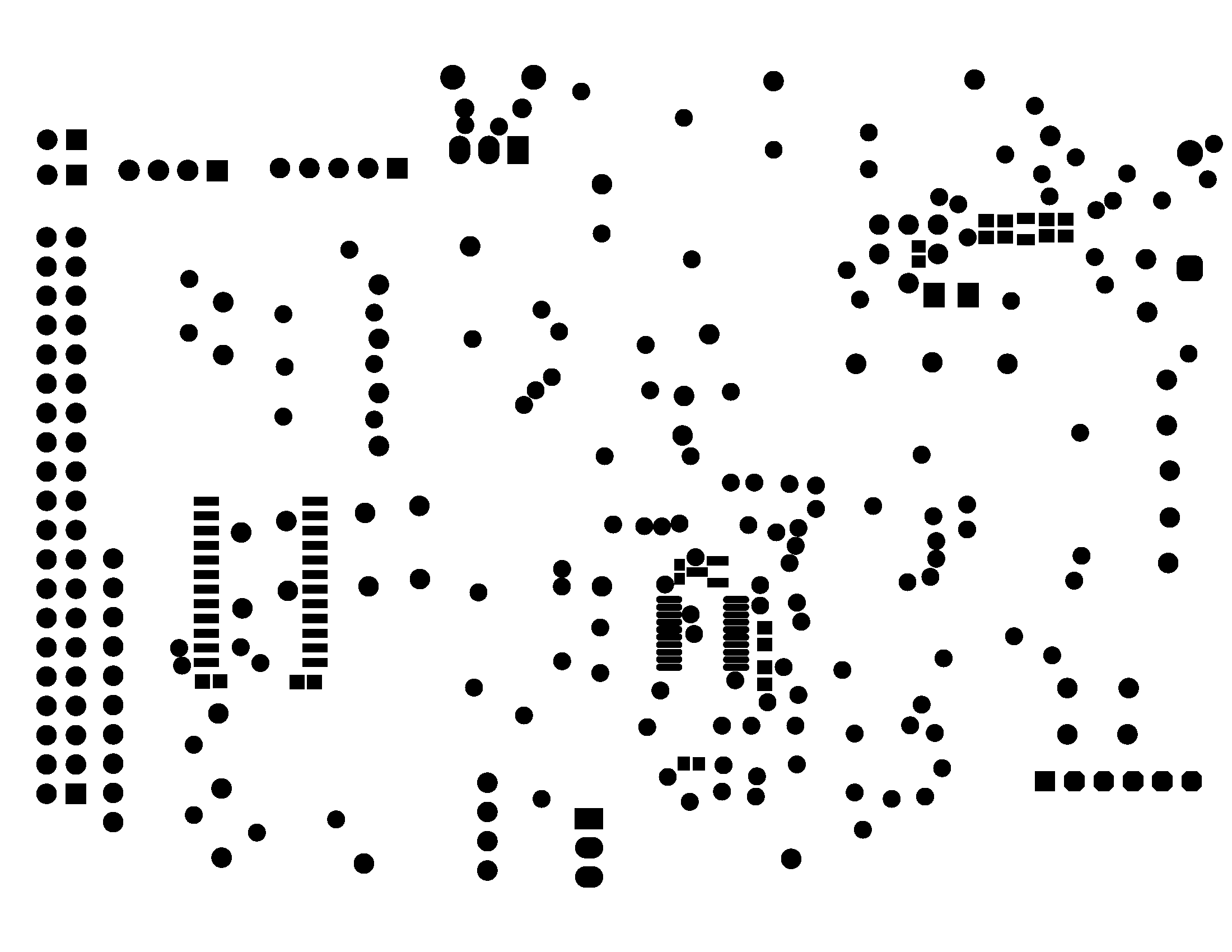


Figure 38 - GraphBit V1.0 : SOLDERMASK BOT

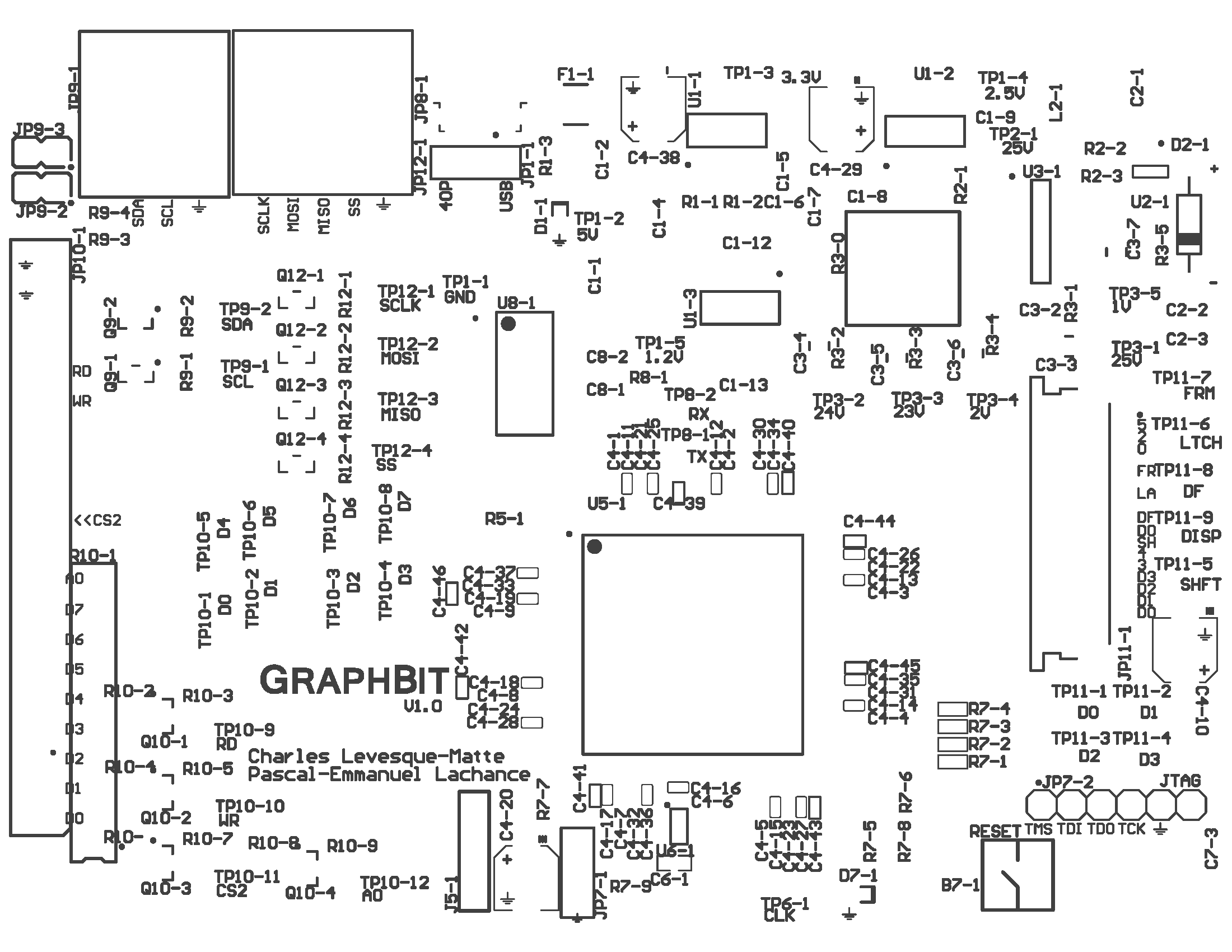


Figure 39 - GraphBit V1.0 : SILKSCREEN TOP

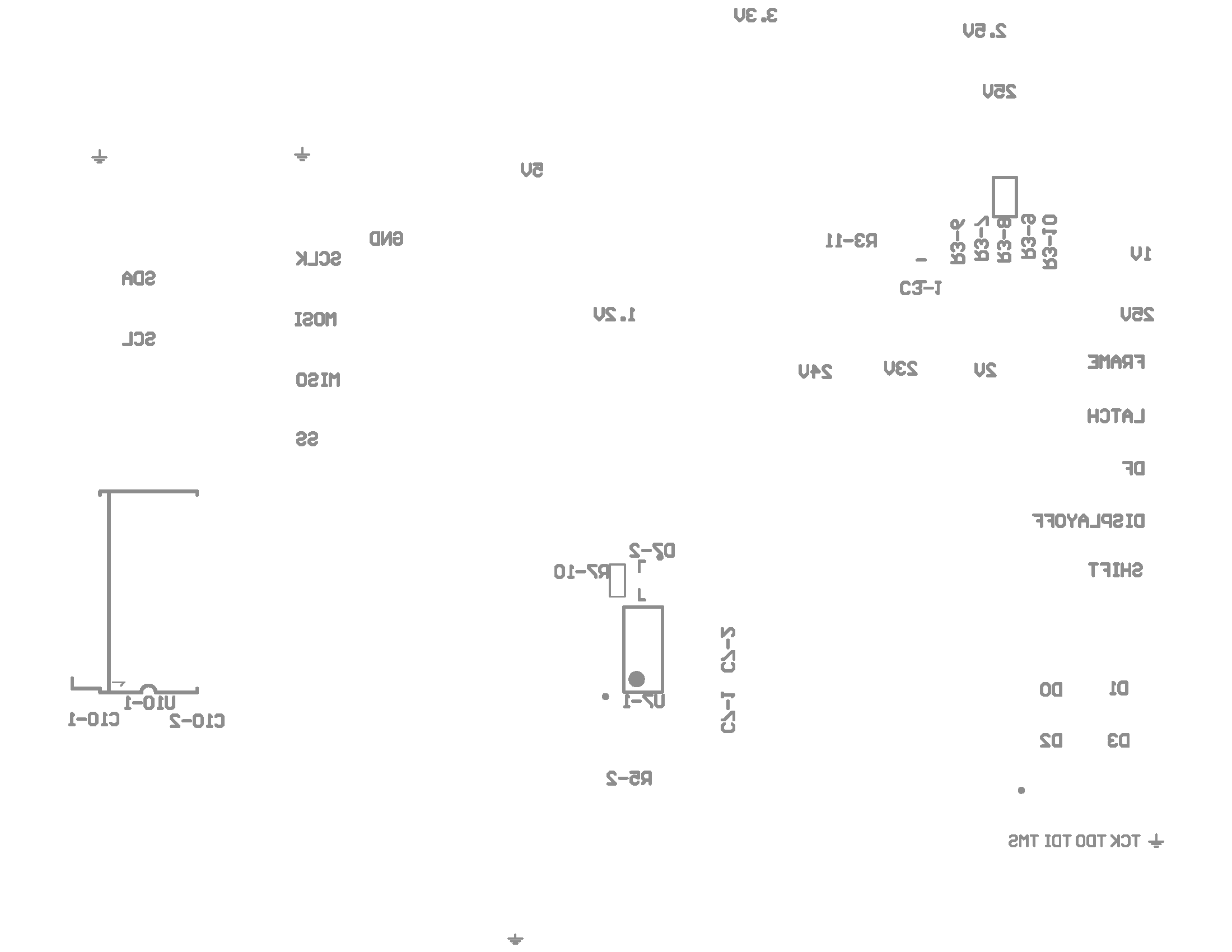


Figure 40 - GraphBit V1.0 : SILKSCREEN BOT

# Schémas V1.1

### Valeur de R2-2

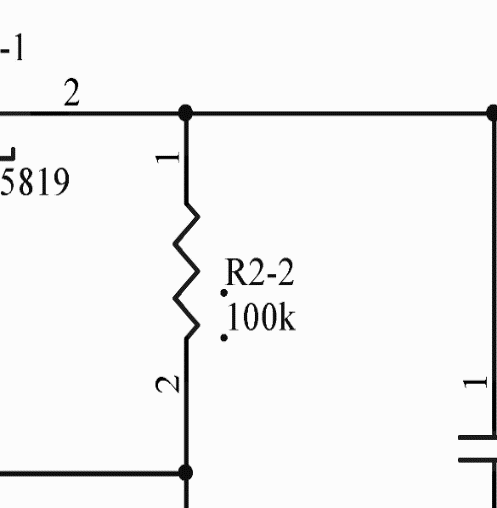
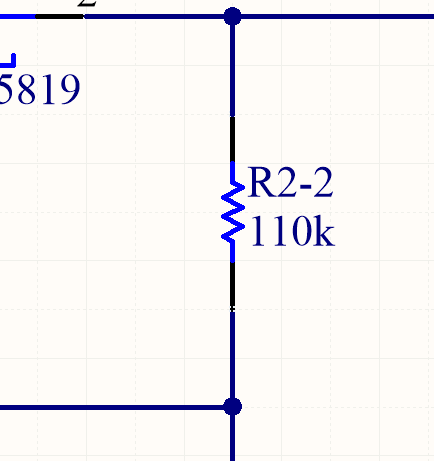


Figure - GraphBit V1.1 Change #1 - R2-2

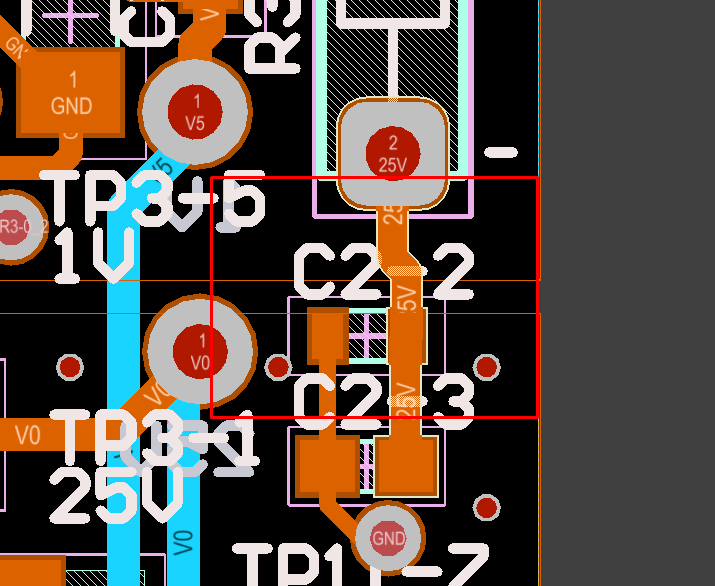
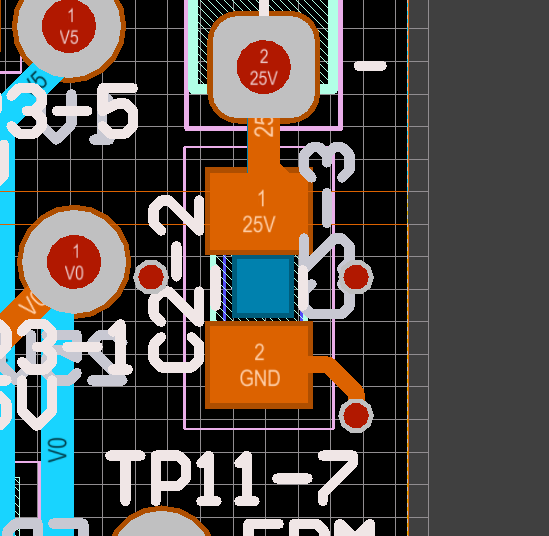
On passe d’une résistance de 100kΩ à une résistance de 110kΩ. Ce changement se produit au niveau du diviseur de tension permettant d’ajuster la sortie du circuit de step-up AP3012, et permet de changer la tension de sortie de 25V à 28V.

Ce patch est nécessaire car notre ampli-op ([OPA4172IDR](https://www.ti.com/lit/ds/symlink/opa172.pdf)) n’est pas réellement rail-to-rail comme sa datasheet l’indique.

### Footprint de C2-2

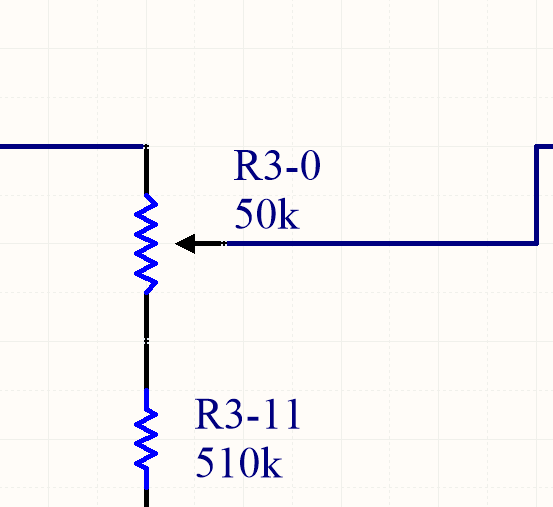
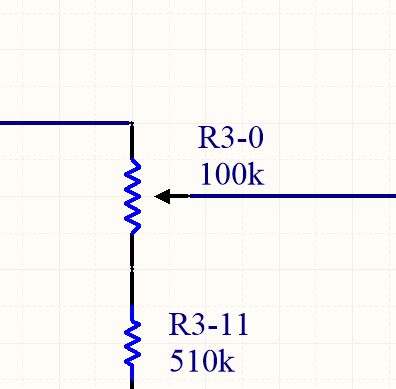
Le condensateur C2-2 est de format 1206, mais notre footprint était de format 0603.

Figure - GraphBit V1.1 Change #2 - C2-2



### Valeur de R3-0

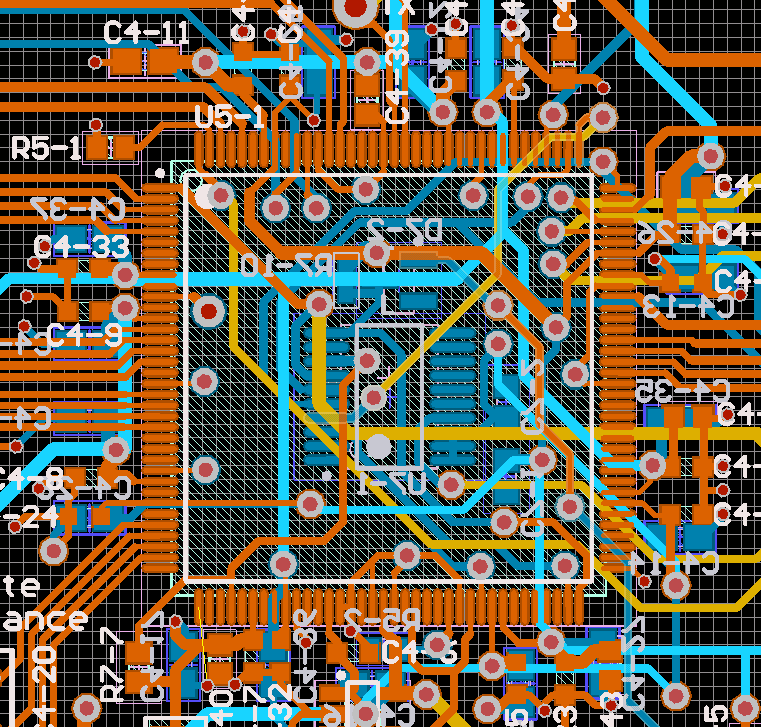
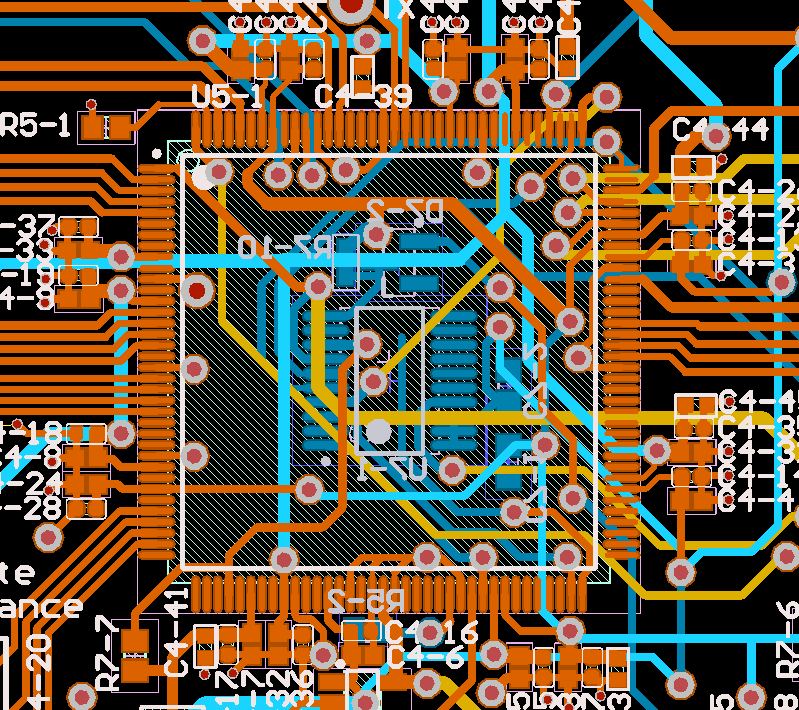
Figure - GraphBit V1.1 Change #3 - R3-0



Nous changeons le potentiomètre de 50kΩ servant à l’ajustement de la tension de polarisation de l’écran pour un potentiomètre de 100kΩ afin d’augmenter la plage de tensions.

### Changer les condensateurs 0402 par des 0603

Figure - GraphBit V1.1 Change #4 - Condensateurs 0402

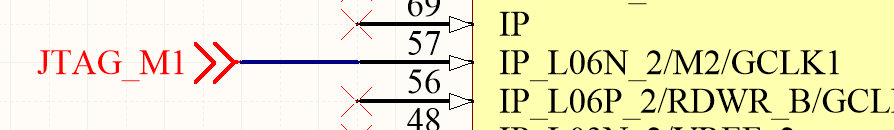
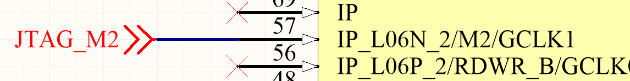


Les condensateurs 0402 sont juste vraiment trop petits et difficile à souder. On cherchait à économiser de l’espace, mais on arrive à un meilleur résultat, plus propre et surtout facile à souder, en employant également la couche du bas pour la disposition des condensateurs. On aurait pu faire avec moitié moins.

### Connecter le signal M2 qui permet de charger la EEPROM

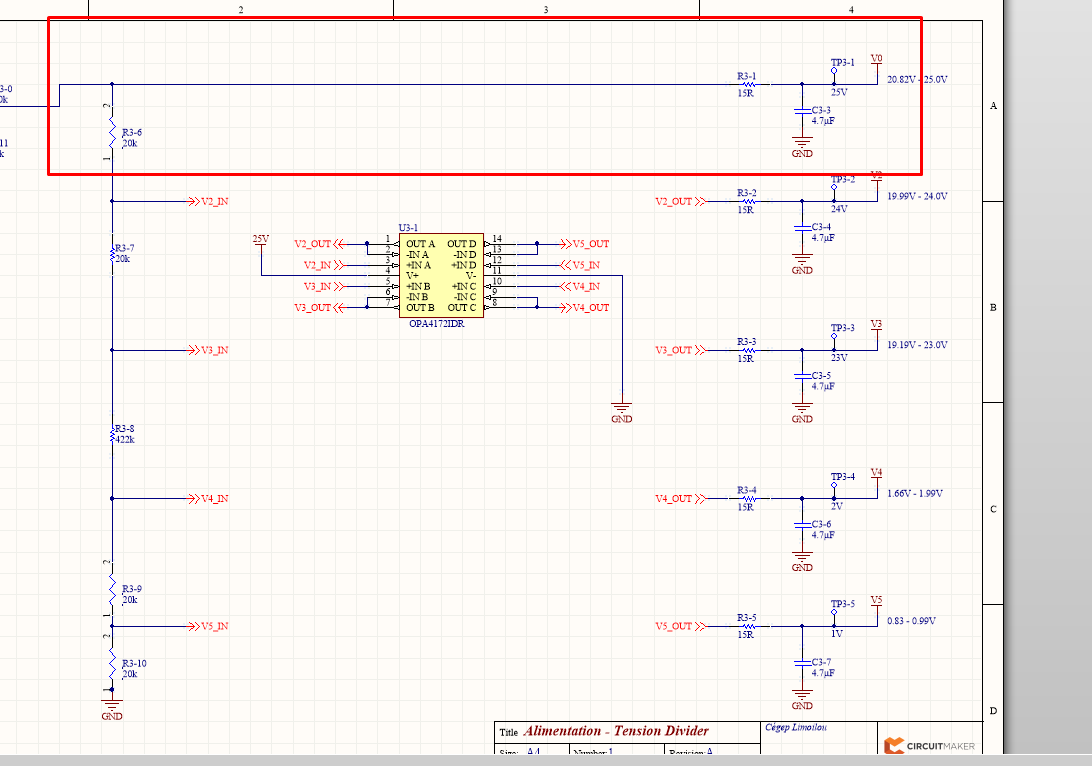
J’ai oublié de renommer le signal M2, donc le signal "M1", inexistant, était connecté à la pin M2, la laissant flottante et empêchant le FPGA d’aller chercher des informations dans la EEPROM.

Figure - GraphBit V1.1 Change #5 - Signal M2



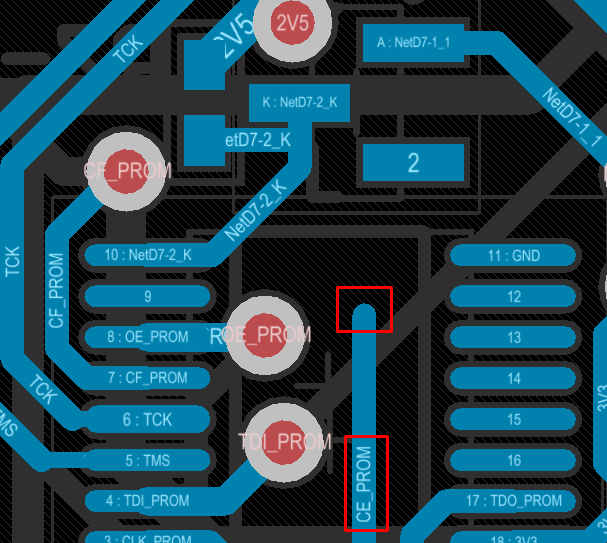
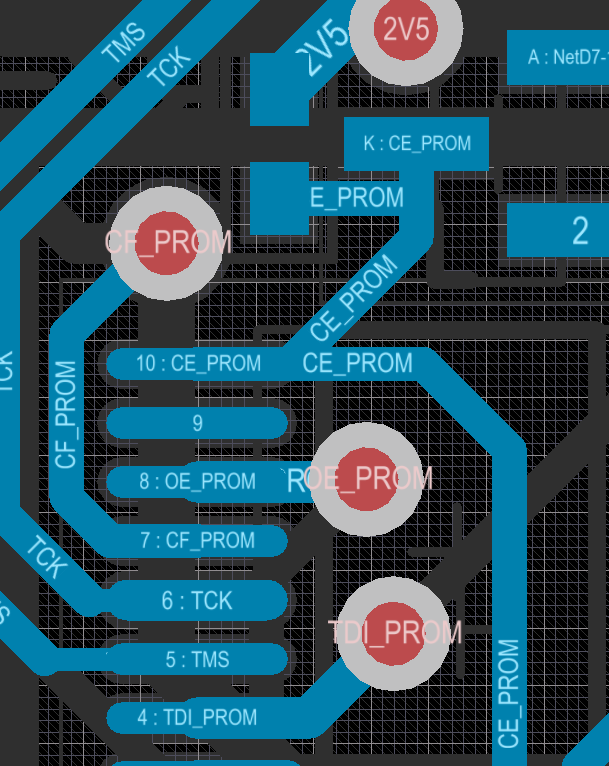
### Passer V0 par ampliop

Figure - GraphBit V1.1 Change #6 - Signal V0



### Connecter le signal CE\_PROM

Figure - GraphBit V1.1 Change #7 - Signal CE\_PROM



Le signal CE\_PROM permet à la EEPROM d’indiquer au FPGA qu’il est prêt à envoyer ses données. J’ai renommé un net à moment donné et puis juste oublié l’existence de ce signal.

### Ajout du UART

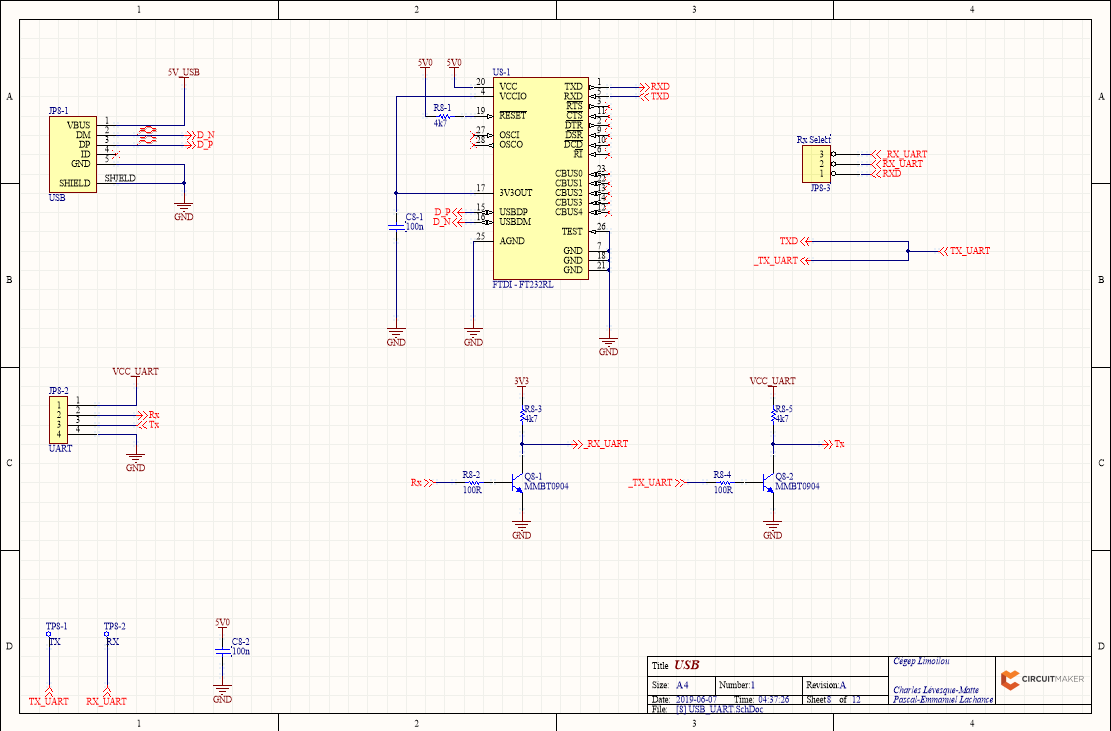


Figure - GraphBit V1.1 : [8] - USB & UART

### Connecteur à l’envers!

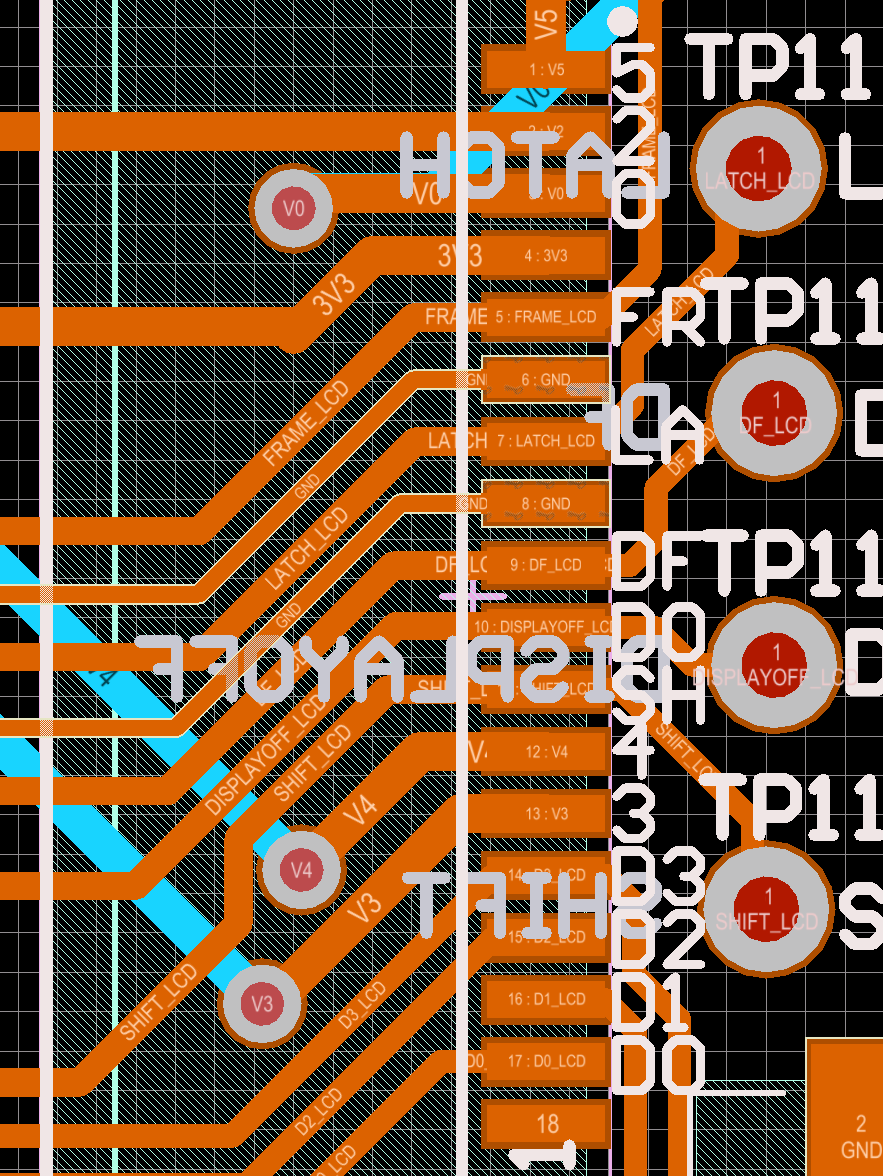


Figure - GraphBit V1.1 Change #9 - Connecteur à l'envers

Le connecteur de l’écran a sa patte 1 en bas, et sa patte 18 en haut. Nous avons fait l’inverse. Cela nous a donné du fil à retordre pendant 2 semaines durant lesquelles nous ne comprenions pas les étranges problèmes que nous avions.

Heureusement, la solution était facile, il s’agissait de tourner le connecteur de 180°, et d’avoir l’écran qui pend dans le vide à droite de la carte, au lieu d’avoir l’écran au-dessus du PCB comme on pensait le faire au début.

# Code VHDL

# Librairie C

## Exemple et Schéma Bloc

Pour l’exemple schéma bloc d’appel de fonction, j’ai choisi le chemin le plus long que le code pouvait prendre, il s’agit de l’écriture de texte en mode parallèle.

#include <string.h>

#include "main.h"

#include "stm32f4xx\_hal.h"

#include "Parallel.h"

#include "LCD.h"

void main(void)

{

const char Message[] = "Voici ce a quoi ressemblerait le code d’un utilisateur de l’ecran qui

cherche a afficher du texte";

// Initialize the parallel module

LCD\_Init\_Parallel(NULL);

// Writes the message on the LCD

LCD\_WriteText(Message, strlen(Message), 1, 0);

}

Après sa vérifications initiale de tous ses paramètres, la fonction LCD\_WriteText() commence par vérifier s’il y a un "\n" dans la string qu’il reçoit. Si oui et qu’il est suffisamment proche, la fonction appellera la fonction LCD\_WriteLine() immédiatement, en lui demandant d’écrire jusqu’au "\n", puis passera à la ligne suivante, et recommencera ce cycle jusqu’à la fin de la string. S’il n’y a pas de "\n" ou qu’il est trop loin, la fonction vérifiera la position du dernier " "(espace), dans la ligne, puis appellera la fonction LCD\_WriteLine() pour écrire sur la ligne jusqu’à l’espace. Ainsi, grâce à cette fonction maison, on ne coupera jamais en plein milieu d’un mot.

size\_t RemainingText = textSize;

uint8\_t currentLine = line;

do

{

size\_t lenght;

char string[LCD\_TEXT\_MAX\_COLUMNS];

/\* strchr, from <string.h>, looks for the first instance of the character '\n' ([LINE FEED], 0x0A) in the

\* string pointed by &pText[(textSize - RemainingText)]. If no instances are found,

\* it returns a null pointer. If the "\n" character is found, it returns a pointer to

\* it's position in the string. \*/

/\* strrchr looks for the last instance of the character ' ' ([SPACE], 0x20) in the

\* string copied in the variable string. \*/

const char\* nextLine = strchr(&pText[(textSize - RemainingText)], '\n');

if (nextLine == NULL) /\* No Line feed in the current string \*/

{

lenght = LCD\_TEXT\_MAX\_COLUMNS - column;

strncpy(string, &pText[(textSize - RemainingText)], lenght);

nextLine = strrchr(string, ' ');

if (nextLine == NULL) /\* No spaces in the current string \*/

{

RemainingText -= (lenght - 0x01U);

WriteText\_Error = LCD\_WriteLine(string, lenght, currentLine, column);

currentLine++;

}

else

{

lenght = (nextLine - string) + 0x01U;

RemainingText -= lenght;

WriteText\_Error = LCD\_WriteLine(string, lenght, currentLine, column);

currentLine++;

}

}

else

{

/\* Get the lenght by taking the position of the '\n' character in memory minus

\* the start position of the string in memory. \*/

lenght = (nextLine - &pText[(textSize - RemainingText)] + 1);

/\* The next "\n" is too far away \*/

if (lenght > (LCD\_TEXT\_MAX\_COLUMNS - column))

{

lenght = LCD\_TEXT\_MAX\_COLUMNS - column;

strncpy(string, &pText[(textSize - RemainingText)], lenght);

nextLine = strrchr(string, 0x20);

if (nextLine == NULL) /\* No spaces in the current string \*/

{

RemainingText -= (lenght - 0x01U);

WriteText\_Error = LCD\_WriteLine(string, lenght, currentLine, column);

currentLine++;

}

else

{

lenght = (nextLine - string) + 0x01U;

RemainingText -= lenght;

WriteText\_Error = LCD\_WriteLine(string, lenght, currentLine, column);

currentLine++;

}

}

else /\* The next '\n' is close enough \*/

{

strncpy(string, &pText[(textSize - RemainingText)], (lenght + 1));

RemainingText -= lenght;

WriteText\_Error = LCD\_WriteLine(string, lenght, currentLine, column);

currentLine++;

}

}

if (WriteText\_Error != NO\_ERROR)

{

break;

}

} while ((RemainingText > 0x0000U) && (RemainingText <= textSize));

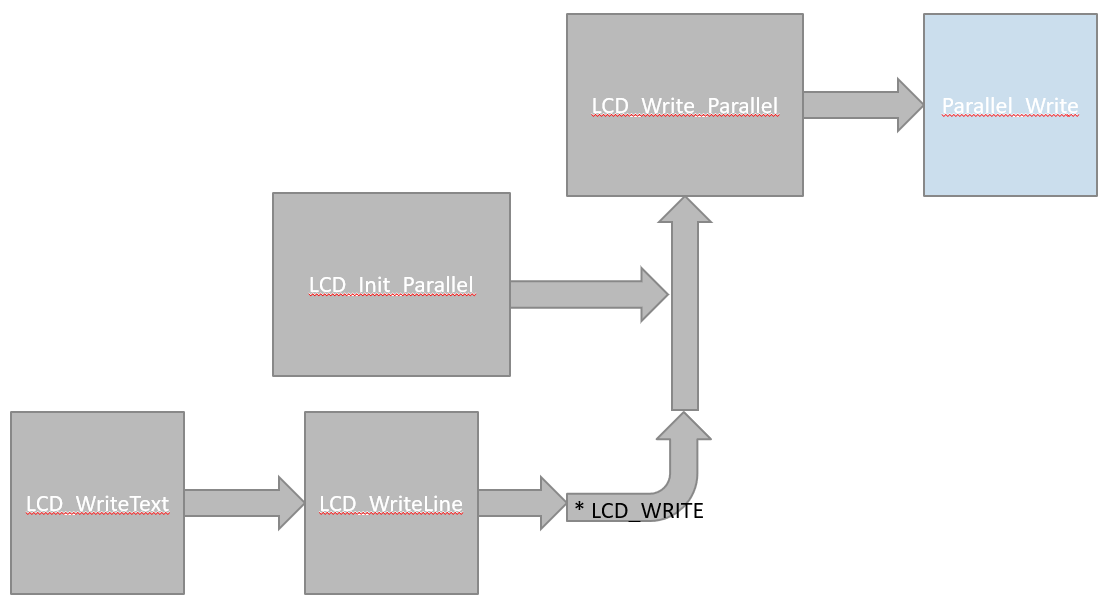
La fonction LCD\_WriteLine() prépare sa string à être envoyée en y ajoutant la ligne et la colonne où la string doit être ajoutée, puis appelle le pointeur de fonction LCD\_Write(), qui, dans le cas du module LCD initialisé en mode parallèle, pointe vers la fonction LCD\_Write\_Parallel(). Cette dernière finit de formatter les données à envoyer vers le FPGA en y ajoutant la taille de la trame, la commande (dans ce cas-ci, texte), et un checksum. Elle appelle aussi les fonctions Parallel\_Write et Parallel\_CS, qui servent respectivement à écrire sur le bus de donnée parallèle, et à contrôler individuellement une pin de chip select.

Figure - Appels de fonction d'une écriture en mode texte

## Liste de variables globales

Pas de variables globales!

Techniquement, une structure, LCD\_Struct, est accessible de façon globale, mais uniquement à l’intérieur du *scope* du fichier .c, et donc n’est pas accessible dans le main ou par qui que ce soit. L’emploi d’une structure semi-globale permet de simuler l’instance d’une classe, car j’aurais préféré faire ce code en C++.

## Conformité MISRA-C :2004

Après avoir travaillé avec les normes MISRA dans le cadre de mon premier stage, j’ai été pris d’une passion pour le code sécuritaire et les normes de programmation, ce qui m’a amené à proposer à mon collègue Charles de faire notre projet conforme aux normes [MISRA-C :2004](http://caxapa.ru/thumbs/468328/misra-c-2004.pdf). L’ensemble des 141 règles de programmations sont respectées dans notre code, sauf à quelques exceptions qui sont levées selon la procédure d’exception de la MISRA, qui permet d’ignorer des règles si on est bien conscient de ce que l’on fait.

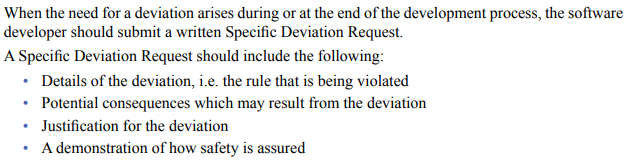


Figure - Guidelines de création d'une specific deviation (MISRA-C:2004)

### Parallel

Dans la librairie Parallèle, les règles 14.6 (*For any iteration statement there shall be at most one break statement used for loop termination.*) et 16.1(*Functions shall not be defined with a variable number of arguments.*) sont brises chacune à 3 reprises, dans chaque fonction d’initialisation avec paramètres variadiques.

/\* 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 \*/

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

\* We're creating a specific deviation for this function, Parallel\_InitData, that can exit the data acquisition loop

\* if an error arises either while receiving the port or the pin.

\* 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.

\* 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 \*/

### LCD

Dans la librairie LCD, les règles 14.6(*For any iteration statement there shall be at most one break statement used for loop termination.*) et 20.4(*Dynamic heap memory allocation shall not be used.*) sont brisées. Techniquement, la règle 20.4 n’est en réalité jamais brisée, car elle était utilisée dans une fonction que n’a finalement jamais eu à servir, qui créait une structure contenant un pointeur vers un tableau de pointeurs de tableaux, et venait mettre un pointeur vers cette structure dans une autre structure. Cela permettait de créer et d’accéder à un tableau bidimensionnel à taille variable, dont la mémoire serait alloué dans la heap.

/\* 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. \*/

/\* MISRA-C : 2004 prohibits the usage of dynamic heap memory allocation (Rule 20.4)

\* We're creating a specific deviation for this function, LCD\_Create\_Array, that not only creates

\* a dynamic array, but creates a dynamic array of dynamic arrays.

\* This is done for optimization reasons for speed, calculation, communication and memory usage.

\* We're checking and double-checking every pointer, and making sure all values are valid before using them.

\* Any doubt of invalidity instantly aborts the creation of the array and calls LCD\_Free\_Array to free allocated memory.

\* We're also freeing the memory after use, in order to prevent memory hogging.\*/

# Programme C#

# Site Web