



وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة عبد الحميد ابن باديس مستغانم



Université Abdelhamid Ibn Badis de Mostaganem

كلية العلوم و التكنولوجيا

Faculté des Sciences et de la Technologie

N°d'ordre : M...../GE/2021

MEMOIRE DE FIN D'ETUDES DE MASTER ACADEMIQUE

FILIERE : TELECOMMUNICATIONS

Spécialité : Système des Télécommunication

Thème

Etude et Implémentation sur FPGA du protocole I2C:
Application au contrôle du codec audio WM8731

Présenté par :

- EDDEBI Amira Souhila
- KADDOUR PACHA Hanane

Soutenu le 12 / 07 / 2021 devant le jury composé de :

Président : M. BENTOUMI Mohamed MCB Université de Mostaganem
Examineur : M. BENBELLA Djelti MCB Université de Mostaganem
Encadreur : M. ABED Mansour MCA Université de Mostaganem
Co-encadreur : M. LARBI BEKLAOUZ Hadj MCB Université de Mostaganem

Année Universitaire : 2020/ 2021

Remerciements

Après avoir remercié Dieu Gloire à Dieu

Nous tenons à remercier toutes les personnes qui ont contribué au succès à notre étude et qui nous ont aidés lors de la rédaction de ce mémoire.

*Nous voudrions dans un premier temps remercier nos encadreurs de mémoire **M.ABED Mansour**, et **M.BEKLAOUZ LARBI Hadj**, qui est aussi notre chef de département de génie électrique, à l'université de Mostaganem pour leurs patience, leurs disponibilité et surtout leurs judicieux conseils, qui ont contribué à alimenter nos réflexions.*

Nous remercions également toute l'équipe pédagogique de la Faculté des Sciences et de la Technologie et tout le staff, qui ont contribué de près ou de loin à cette réussite.

Nous tenons à exprimer ma gratitude aux personnes qui m'ont aidé à produire ce mémoire. Ceux qui ont partagé avec nous leur expérience et leurs connaissances dans ce domaine, tout en nous accordaient leur confiance et leur grande indépendance dans l'exécution de missions valorisantes.

A titre personnel

Nous remercions encore une fois nos familles, pour leur soutien constant et leurs encouragements.

Enfin

Nous espérons que ce mémoire sera encourageant et bénéfique pour les étudiants des années prochaines, j'espère que vous prendrez plaisir à le lire.

Dédicace

Je dédie ce travail

A mes chères parents, pour tous leur sacrifices, leur amour, leur tendresse, leur soutien et leur prières tout au long de mes études,

A tous mes chers membres de familles, pour leur appui et leur encouragement,

A mes chers professeurs, En raison de leur sérieux au travail, de leur empressement à nous apprendre et leur encouragement,

Et aussi aux étudiants des années précédentes et prochaines, et j'espère qu'ils seront une référence pour tout développement futur.

EDDEBI AMIRA SOUHILA

Dédicaces

*Je dédie ce travail avec beaucoup de remerciements
et d'appréciation à :*

Ma mère la plus importante personne dans ma vie,

*A Mes encadrateurs dont j'ai trouvé un grand plaisir
à travailler avec : M. Abed Mansour et
M. Larbi Beklaouz Hadj*

*A Mes frères qui ont cru en mes capacités
Et le plus grand dédicace A Anwar Walid qui a eu
un grand rôle dans ce travail*

Merci infiniment

KaddourPacha Hanane

Résumé

Afin de mettre en œuvre correctement le protocole de communication I2C pour le contrôle des signaux audio, dans ce mémoire, nous avons d'abord commencé par une explication générale et exemples concernant les différents protocoles et normes utilisés dans divers domaines de l'électronique. Après, nous avons discuté le protocole I2C en particulier, en termes de longueur de paquet, transmission de données, adresses et utilisation. Nous avons finalement mis en place un protocole de communication série, dans notre cas, le protocole I2C dans une carte FPGA de référence ALTERA Cyclone II. La première application pratique de cette implémentation est le contrôle du signal sonore.

Pour se faire, nous avons utilisé la carte de développement DE1 comprenant la FPGA ALTERA Cyclone II qui contrôlera le codec audio WM8731 embarqué sur la même carte de développement. Il est à noter que tous les codes sources ont été développés en langage VHDL en utilisant le logiciel Quartus ALTERA V 13.

ملخص

من أجل التنفيذ الصحيح لبروتوكول الاتصال I2C للتحكم في الإشارات الصوتية ، بدأنا في هذه المذكرة أولاً بشرح عام و تقديم أمثلة بخصوص البروتوكولات والمعايير المختلفة المستخدمة في مختلف مجالات الإلكترونيات. بعد ذلك، تطرقنا إلى بروتوكول الاتصال I2C على وجه الخصوص، من حيث طول الحزمة ونقل البيانات والعناوين والاستخدامات الممكنة لهذا البروتوكول. أخيراً طبقنا بروتوكول اتصال تسلسلي، في حالتنا، بروتوكول I2C في لوحة رقمية مبرمجة من نوع FPGA ALTERA Cyclone II. أول تطبيق عملي لهذا التنفيذ المادي هو التحكم في الإشارة الصوتية للقيام بذلك، استخدمنا لوحة التطوير DE1 المحتوية على ALTERA Cyclone II والتي ستنحكم في وحدة ترميز الصوت WM8731 المضمن في نفس لوحة التطوير. تجدر الإشارة إلى أنه تم تطوير جميع البرمجيات بلغة VHDL باستخدام برنامج Quartus ALTERA V 13.

Summary

In order to properly implement the I2C communication protocol for audio signals' control, in this thesis, we have first started with a general explanation and examples concerning different protocols and standards used in various electronics fields. Then, we have discussed the I2C protocol in particular, in terms of packet length, data transmission, addresses and usage. At the end, we have implemented a serial communication protocol, in our case, the I2C protocol in an FPGA card of reference ALTERA Cyclone II. The first practical application of this implementation is the control of the sound signal.

To realize our project, we have used the DE1 development board including the ALTERA Cyclone II FPGA that will control the audio codec WM8731 embedded at the same development board. Note that all source codes have been developed in VHDL language using Quartus ALTERA V 13 software package.

Liste des abréviations

ADC	Analog to Digital Converter
CAN	Control Area Network
CC	Courant continu
CEC	contrôle entre appareils
CEI	Commission Electrotechnical International
CPU	Central Processing Unit
CSI	Camera Serial Interface
CTS	Clear to send
DAC	Digital to Analog Converter
DCD	Data carier detected
DDC	le signal de synchronisation
DDWG	Digital Display Working Group
DSI	Display Serial Interface
DSR	DCE Ready
DTR	DTE Ready
DVD	Digital Video Disc
DVI	déigital vésuel interface
EPROMs	erasable programmable read-only memory
FPD-Link	Flat Panel Display Link
FPGA	Field Programmable Gate Arrays
GND	Signal Ground
HART	Highway Addressable Remote Transducer
HDMI	High-Definition Multimedia Interface
Hi-Fi	high fidelity
IEEE	Institute of Electrical and Electronics Engineers
I2C	INTER INTEGRATED CIRCUIT
IO-Link	Inter Out Link
ISO	International Organization for Standardization
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LHPOUT	Left Channel Headphone Output
LIN	Local interconnect network
LLINEIN	Left Channel Line Input

LVDS	Low-voltage differential signalling
PC	ordinateur industriel
PCB	Printed Circuit Board
MIPI	Mobile Industry Processor Interface
M-LVDS	Multipoint Low-voltage differential signalling
MSB	Most Significant Bit
PCI	Peripheral Component Interconnect
PCI E	Peripheral Component Interconnect Express
RHPOUT	Right Channel Headphone Output
RLINEIN	Right Channel Line Input
RMS	root mean square
RS	Recommended Standard
RTS	Request to send
Rx	Receive data
R/W	Read/write
SAS	Serial Attached SCSI
SATA	Serial Advanced Technology Attachment
SCL	Serial Clock
SCSI	Small Computer System Interface
SDA	Serial Data
SoC	Security Operation Center
SPI	Serial Peripheral Interface
TCP/IP	Transport control protocol/ internet protocol
TIA/EIA	Telecommunications Industries Association/Electronic Industries Association
TMDS	transfert du signal audio et vidéo
TTL	Transistor transistor logic
Tx	Transmit data
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal serial bus
VHDL	VHSIC Hardware Description Language

Liste des Figures

Chapitre 1

Figure 1.1 : Liaison LVDS point à point [5].	4
Figure 1.2: Bus M-LVDS Half-Duplex [4].	5
Figure 1.3: Bus M-LVDS Full-Duplex[4].	5
Figure 1.4: Schéma d'application typique de l'interface FPD-Link [7].	5
Figure 1.5 : Bus CAN [9].	6
Figure 1.6 : Formats de trame de message du bus CAN [11].	7
Figure 1.7: Le réseau CAN automobile [12].	8
Figure 1.8: bus I2C [13].	8
Figure 1.9 : Topologie du bus LIN [14].	9
Figure 1.10: Trame de bus LIN [15].	9
Figure 1.11 : Exemple d'usage du protocole LIN [16].	9
Figure 1.12 : Types de connecteurs DVI femelles [17].	10
Figure 1.13: Trame de donnée du protocole DVI avec FPGA [19].	11
Figure 1.14 : Différents types de connecteurs HDMI [22].	11
Figure 1.15 : Prise HDMI [24].	12
Tableau 1.3 : Comparaisons entre les ports Display Port [23].	13
Figure 1.16 : Paquet de donnée Display-Port [26].	13
Figure 1.17 : Schéma des connecteurs Display-Port [27].	14
Figure 1.18 : Mode multipoint et point à point configuration réseau [29].	14
Figure 1.19: Trame de message HART [30].	15
Figure 1.20 : Paquet UART [31].	16
Figure 1.21 : Schéma bloc d'une architecture PCI [32].	16
Figure 1.22 : Exemple de transfert du brut data [34].	17
Figure 1.23 : Quatre emplacements du PCIe standard [37].	17
Figure 1.24 : Détection de vitesse USB [38].	18
Figure 1.25 : La trame USB [39].	18
Figure 1.26 : Schéma fonctionnel d'un appareil mobile typique [41].	19
Figure 1.27 : Bus CSI du Raspberry PI [42].	19
Figure 1.28 : L'interface MIPI-DSI [44].	20
Figure 1.29: Exemple de l'interface DSI [44].	20
Figure 1.30: Niveau de paquet (paquets courts et longs) [45].	20
Figure 1.31: Implémentation SPI maître unique, esclave unique [47].	21

Figure 1.32: Mises en œuvre de plusieurs esclaves SPI [47].	21
Figure 1.33 : Trame de donnée SATA [48].	22
Figure 1.34: Câbles SATA [48].	22
Figure 1.35 : Adresse SAS [51].	23
Figure 1.36 : Séquence de communication maître- Device IO-Link [53].	24
Figure 1.37: Bus Ethernet [55].	24
Figure 1.38 : Paquet Ethernet [54].	24
Figure 1.39 : Le modèle de communication du TCP/IP.	25
Chapitre 2	
Figure 2.1 : bus I2C	27
Figure 2.2 : Structure d'E/S d'un module I2C	28
Figure 2.3 : Câblage I2C	29
Figure 2.4 : Condition de départ et d'arrêt	29
Figure 2.5 : Exemple de transmission réussie	30
Figure 2.6 : Trame de base : Ecriture d un octet	31
Figure 2.7 : Exemple d'octet d'adresse	32
Figure 2.8 : Exemple de lecture d'une donnée	33
Figure 2.9 : Trame de base : Etat d'attente	33
Chapitre 3	
Figure 3.1 : Schéma fonctionnel du codec audio WM8731.	35
Figure 3.2: Connexion duWM8731 à un circuit programmable : CPU, FPGA, µcontrôleur	37
Figure 3.4 : Contrôleur I2C	38
Figure 3.3: Interface audio numérique avec FPGA Cyclone II	39
Figure 3.5 : Configuration matérielle de la méthode de contrôle de son réalisé	40
Figure 3.6 : L'environnement de développement Quartus II.	41
Figure 3.7 : Création d'un nouveau projet sous Quartus II.	42
Figure 3.8: Le circuit logique de l'audio_PLL.	43
Figure 3.9: data rom.	43
Figure 3.10 : Codec_WM8731.	43
Figure 3.11: Câblage I2C.	44
Figure 3.12 : Adresse du registre LLINEIN.	44
Figure 3.13: Schéma global de la FPGA et le codec audio WM8731.	45
Figure 3.14 : Photo représentant les manipulations.	45
Figure 3.15: Envois du code source à l'DE1 après montage des différents composants et connectics.	46

Liste des tableaux

Chapitre 1

Tableau 1.1: Comparaison entre CAN Low-speed et CAN High-speed [9].	7
Tableau 1.2 : Comparaisons entre les ports HDMI [23].	12
Tableau 1.4: Les principales différences entre les interfaces RS-232, RS-422 et RS-485 [32].	15
Tableau 1.5 : Caractéristiques du bus PCI [33].	16
Tableau 1.6 : Vitesses de transmission maximales selon les différentes normes USB [40].	18
Tableau 1.7 : Description des signaux [47].	21
Tableau 1.8 : Taux de liaison physique par direction [50].	23

Chapitre 2

Tableau 2.1 : les caractéristique de bus I2C	27
Tableau 2.2 : Glossaire I2C	28

Chapitre 3

Tableau 3.1 : Nom, type et description de chaque pin du WM8731	36
Tableau 3.2: Left and Right Channel Line Input (LLINEIN & RLINEIN).	44
Tableau 3.3: Left and Right Channel Headphone Output (LHPOUT & RHPOUT).	44

Sommaire

Liste des abréviations	i
Liste des figures	iii
Liste des tableaux	v
Introduction general	1
<i>Chapitre 1: protocoles de communication</i>	
I.1 Introduction	3
I.2 Définition d'un protocole de communication	3
I.2.1 LVDS (Low-Voltage Differential Signaling)	3
I.2.2 M-LVDS (Multipoint LVDS)	4
I.2.3 FPD-Link (Flat Panel Display Link)	5
I.2.4 CAN (Control Area Network)	6
I.2.5 I2C (INTER INTEGRATED CIRCUIT)	8
I.2.6 LIN (Local interconnect network)	8
I.2.7 DVI (Digital Visuel Interface)	10
I.2.8 HDMI (High-Definition Multimedia Interface)	11
I.2.9 Display-Port	12
I.2.10 HART (Highway Addressable Remote Transducer)	14
I.2.11 UART (Universal Asynchronous Receiver/Transmitter)	15
I.2.12 PCI (Peripheral Component Interconnect) et PCI E (PCI Express)	16
I.2.13 USB (Universal serial bus)	17
I.2.14 CSI et DSI	18
I.2.15 SPI (Serial Peripheral Interface)	21
I.2.16 SAS (Serial Attached SCSI), SATA (Serial Advanced Technology Attachment)	22
I.2.17 IO-LINK	23
I.2.18 Ethernet	24
I.3 Conclusion	25
<i>Chapitre 2: protocole de communication I2C</i>	
II.1 Introduction	26
II.2 Le bus de communication I2C	26
II.2.1 Etude du bus I2C	27
II.3 le protocole I2C	28
II.3.1 Glossaire I2C	28
II.3.2 cablage I2C	29
II.3.3 La prise de contrôle du bus I2C	29

Sommaire

II.3.4 Ecriture d'un octet (une donnée)	30
II.3.5 Transmission d'une adresse	31
II.3.6 Lecture d'une donnée	32
II.4 Utilisation de l'I2C	33
II.5 Conclusion	34
<i>Chapitre 3 : Implémentation matérielle du protocole I2C sur ALRETA Cyclone II : Application au contrôle du codec audio WM8731</i>	
III.1 Introduction	35
III.2 codec audio et communication I2C	35
III.2.1 Codec audio	35
III.2.1.1 Description des pins	36
III.2.1.2 codec audio et circuit programmable	36
III.2.2 Communication I2C	38
III.2.3 Adresse de chip WM8731	40
III.3 Simulations et implémentations sur FPGA	40
III.3.1 présentation de Quartus II	41
II.3.2 Les étapes de simulation	42
II.4 Manipulations sur le projet réalisé	45
III.5 Conclusion	46
<i>Conclusions et recommandations pour les travaux futurs</i>	
I. Conclusions	47
II. Recommandations	48
Bibliographie	49
Annex	
Annex A	55
Annex B	72

Introduction générale

Le domaine de télécommunication est couvert de différentes technologies et disciplines scientifiques. En outre, la communication électronique présente un rôle principal pour évaluer ce domaine. L'objectif est de transmettre des données entre deux ou plusieurs systèmes de telle sorte de reproduire les informations communiquées avec la plus grande fidélité. Pour que les systèmes communiquent entre eux, ils ont besoin d'un ensemble de règles spécifiques à chaque type d'échange de données nommées « protocoles de communication ».

Un protocole de communication peut être défini comme un ensemble de règles standardisées organisant la communication à distance entre systèmes. Cette communication peut s'effectuer en série ou en parallèle ; en mode synchrone ou asynchrone.

La communication série asynchrone signifie qu'un conducteur unique (une seule ligne) est utilisée pour transmettre les données. Si elle est synchrone alors l'acheminement de l'information nécessite l'utilisation de deux lignes : une pour les données et la deuxième pour l'horloge. D'autre part, transmettre l'information bit par bit présente l'avantage de la haute résistance au bruit, la possibilité de communiquer des données sur de longues distances et du coût faible en termes de nombre de fils utilisés.

La communication parallèle, quant à elle, nécessite plusieurs lignes pour transmettre les données sur des paquets de n bits à la fois. Son avantage est que la vitesse de transmission est très élevée, bien que sur des distances limitées. Lorsque la distance augmente, la perturbation bruit produite en raison des amplitudes de la ligne de transmission entraîne une déformation du signal reçu en plus du retard dû à la propagation.

Dans notre thème de mémoire, nous allons étudier et implémenter un protocole de communication série, en occurrence, le protocole I2C (Inter Integrated Circuit), dans une carte FPGA. Cette implémentation a pour application d'établir une communication avec le codec audio.

Parmi les différents choix, nous utilisons la carte FPGA de type ALTERA Cyclone II tandis que le codec audio utilisé est de référence WM8731. Ces deux modules sont intégrés sur la carte de développement DE1. Notre objectif principal est de réaliser le circuit électronique du protocole I2C entre la FPGA et le WM8731 sur la carte DE1 programmé avec le langage VHDL de Quartus ALTERA V 13.0. Le cahier de charge consiste à pouvoir contrôler plusieurs paramètres du codec audio à savoir :

- Volume +/-
- Muter/écouter
- Démarrer/arrêter

Le reste du mémoire est organisé comme suit :

Dans le premier chapitre, nous donnons un aperçu global sur les protocoles de communication les plus utilisés en pratique. Un état de l'art est établi présentant pour chaque protocole étudié sa définition, la structure de son paquet de donnée ainsi que son utilisation pratique,

Le deuxième chapitre est dédié exclusivement au protocole I2C : structure, modes de transmission, câblage, paquets de données ainsi que les avantages et les inconvénients de ce type de protocoles adopté dans ce travail,

Le troisième et le dernier chapitre est réservé au côté pratique, en particulier l'implémentation matérielle du protocole I2C sur ALTERA Cyclone II en utilisant le langage VHDL de Quartus ALTERA V 13.0 en présentant les résultats du contrôle du codec audio conformément au cahier de charge déclaré au début de ce projet.

Ce document est clôturé par des conclusions et des recommandations pour améliorer le présent travail et le développer davantage. Ceci permettra de réaliser des travaux pratiques de haut niveau pour les étudiants de la filière télécommunications, en particulier, en ce qui concerne les technologies et protocoles multimédia. Une annexe est fournie à la fin du document incluant tous les codes sources VHDL utilisés dans l'implémentation proposée.

I.1 Introduction

Pour permettre aux deux systèmes de dialoguer entre eux est l'objectif principal de la communication et de l'échange d'informations. Ces deux systèmes doivent parler le même langage, c'est pourquoi nous avons fixé les règles de communication. Toutes ces règles constituent le protocole de communication.

La réalisation du modèle d'échange d'informations ainsi que l'interface de communication des réseaux électriques intelligents dépendent en grande partie du protocole de communication choisi. L'objectif principal de ce chapitre est d'introduire une vue d'ensemble sur les principaux types de ces protocoles ainsi que leurs utilisations pratiques.

I.2 Définition d'un protocole de communication

Un protocole de communication est un ensemble de règles qui rendent les communications possibles, car les intervenants sont censés les respecter [1].

Nous avons initialement nommé le protocole celui qui est utilisé pour communiquer sur la même couche d'abstraction entre deux appareils différents. Grâce à l'extension du langage, parfois on utilise aussi ce mot pour préciser les règles de communication entre deux couches sur un même appareil. Il existe plusieurs protocoles et normes de communication, chaque protocole ou norme a des caractéristiques distinctives.

I.2.1 LVDS (Low-Voltage Differential Signaling)

LVDS est une norme technique qui spécifie les caractéristiques électriques d'une norme de signalisation différentielle et série, mais ce n'est pas un protocole [2]. Le LVDS fonctionne à faible consommation et peut fonctionner à des vitesses très élevées (jusqu'à plusieurs GBit/s [3]) à l'aide de câbles en cuivre (distances allant jusqu'à 10 m [3]) à paires torsadées peu coûteux. LVDS est une spécification de couche physique uniquement. La norme principale pour le LVDS est TIA/EIA-644. Une autre norme parfois utilisée pour LVDS est IEEE 1596.3—SCI [4].

En 1994, Il à été introduit par National Semi-Conductor, qui est devenu un standard réaliste pour la transmission de données à haut débit [5].

Une liaison LVDS point à point, illustrée à la **Figure 1.1**, se compose d'un émetteur de courant générant une chute de tension aux bornes d'une résistance de terminaison (leur valeur est 100Ω) placée du côté récepteur. La tension (égale 350 mV) aux bornes du la résistance de terminaison est proportionnelle au courant d'entraînement (égale 3.5 mA) et permet au récepteur de reconnaître un signal valide [5].

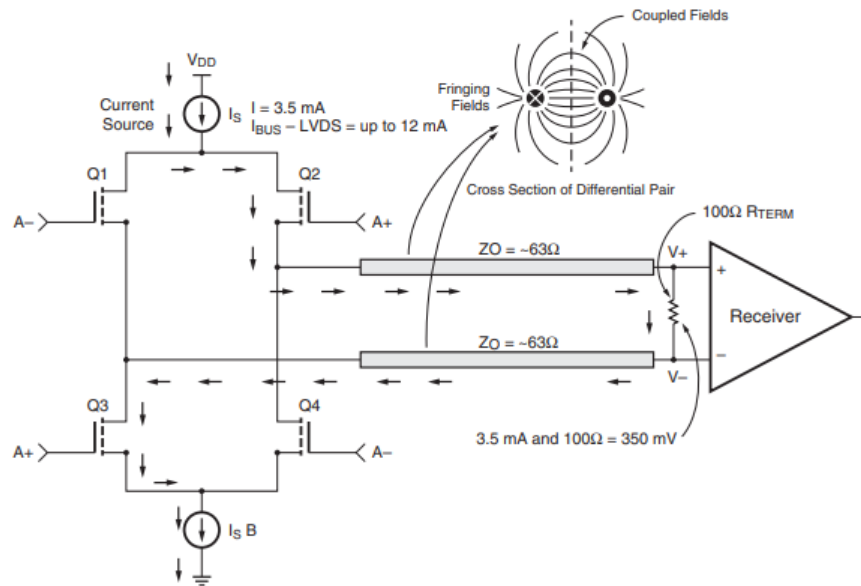


Figure 1.1 : Liaison LVDS point à point [5].

Il existe de nombreuses applications dans de nombreux segments de marché qui utilisent LVDS pour la transmission de données telle que téléviseurs LCD, les systèmes d'infodivertissement automobile, les caméras industrielles et la vision industrielle, les ordinateurs portables et tablettes et les systèmes de communication. Les applications typiques sont la vidéo haute vitesse, les graphiques, les transferts de données de caméra vidéo et les bus informatiques à usage général [2].

1.2.2 M-LVDS (Multipoint LVDS)

La norme M-LVDS (Standard TIA/EIA-899) est une nouvelle norme qui a été créée en réponse à une demande de la communauté des communications de données pour une norme d'interface équilibrée à haute vitesse à utiliser en général pour les applications multipoints [6].

Elle spécifie les pilotes et les récepteurs de signalisation différentielle basse tension pour l'échange de données à travers structures de bus de données semi-duplex ou multipoint. M-LVDS est capable de fonctionner à des taux de signalisation jusqu'à 500 Mbit/s. [6]

Il existe deux types de multipoints bus, semi-duplex et duplex intégral, illustrés à la **Figure 1.2** et **Figure 1.3**, respectivement. Dans un bus semi-duplex, deux fils sont utilisés de telle sorte qu'un appareil puisse transmettre et que les autres appareils puissent recevoir. Dans un bus full-duplex, quatre fils sont utilisés, permettant à un nœud pour transmettre simultanément à un autre émetteur [4].

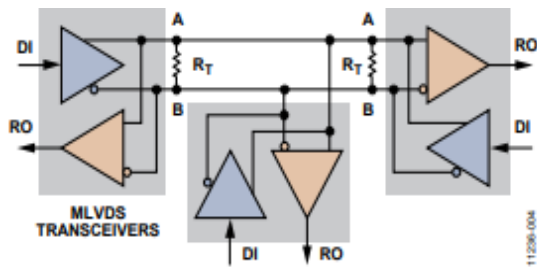


Figure 1.2: Bus M-LVDS Half-Duplex [4].

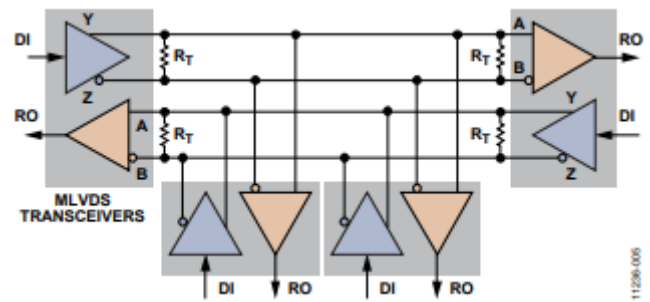


Figure 1.3 : Bus M-LVDS Full-Duplex[4].

I.2.3 FPD-Link (Flat Panel Display Link)

Flat Panel Display (FPD) est un protocole de transmission de flux vidéo numériques des processeurs graphiques vers les écrans numériques. Il s'agit d'une interface interne qui utilise une signalisation différentielle basse tension pour transmettre des paires de bits vidéo sérialisés. Pour ce faire, il sérialise les données TTL parallèles des cartes graphiques. En utilisant trois paires de données et une paire pour l'horloge, le protocole transmet une vidéo RVB 18 bits. Les nouvelles liaisons FPD utilisent quatre paires de données et une paire d'horloges pour transmettre des couleurs 24 bits et 30 bits [7].

Exemple d'application : Le DS99R421 (illustrée à la **Figure 1.4**) convertit une entrée FPD-Link avec 4 LVDS équilibrés non CC (3 données LVDS + horloge LVDS) plus 3 bits de contrôle basse vitesse suréchantillonnés en un seul flux série LVDS équilibré CC avec des informations d'horloge intégrées. Ce flux série unique simplifie le transfert du bus 24 bits sur une seule paire différentielle de pistes et de câbles PCB en éliminant les problèmes d'asymétrie entre les 3 entrées de données LVDS parallèles et les chemins d'horloge VLDS. Il permet de réduire les coûts du système en réduisant 4 paires LVDS à 1 paire LVDS, ce qui réduit à son tour les couches de circuits imprimés, la largeur du câble, la taille du connecteur et les broches [7].

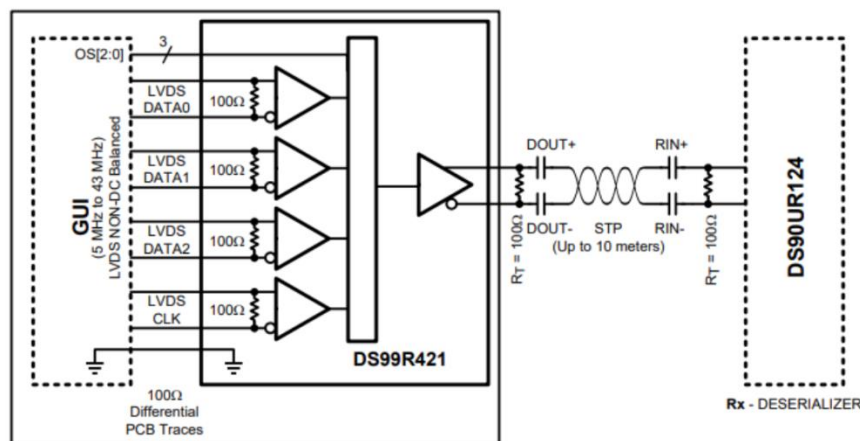


Figure 1.4: Schéma d'application typique de l'interface FPD-Link [7].

FPD-Link II est une version améliorée de FPD-Link qui a été introduite en 2006. Elle était spécialement conçue pour les applications automobiles et les interfaces de caméra. FPD-Link III était introduite en 2010 pour réduire les câbles. Elle est particulièrement utilisée pour se débarrasser des câbles pour les canaux de contrôle tels que les bus I2C et CAN [8].

1.2.4 CAN (Control Area Network) [9]

C'est un protocole (réseau) de communication série asynchrone développé par Robert Bosch (fournisseur de système automobile allemand en 1985) [10]. Le CAN était normalisé par la norme ISO11898 pour les applications grandes vitesses et ISO11519 pour les applications basses vitesses. [10]

Le CAN est un bus multi-maitre, chaque système relié avec le bus est appelé un nœud. Ces nœuds sont d'un nombre illimité et sans adresses. Cela signifie que tout les nœuds se considèrent égaux, et donc peuvent tous envoyer en même temps. Pour cette raison, un « filtrage de message » doit être effectué pour déterminer la priorité et les données sur le bus concerné.

La **Figure 1.5** montre que ce bus a deux lignes (le câble contient une paire torsadée blindée ou non blindée) avec une résistance de polarisation Il existe deux types de transmission: Low-speed et High-speed.

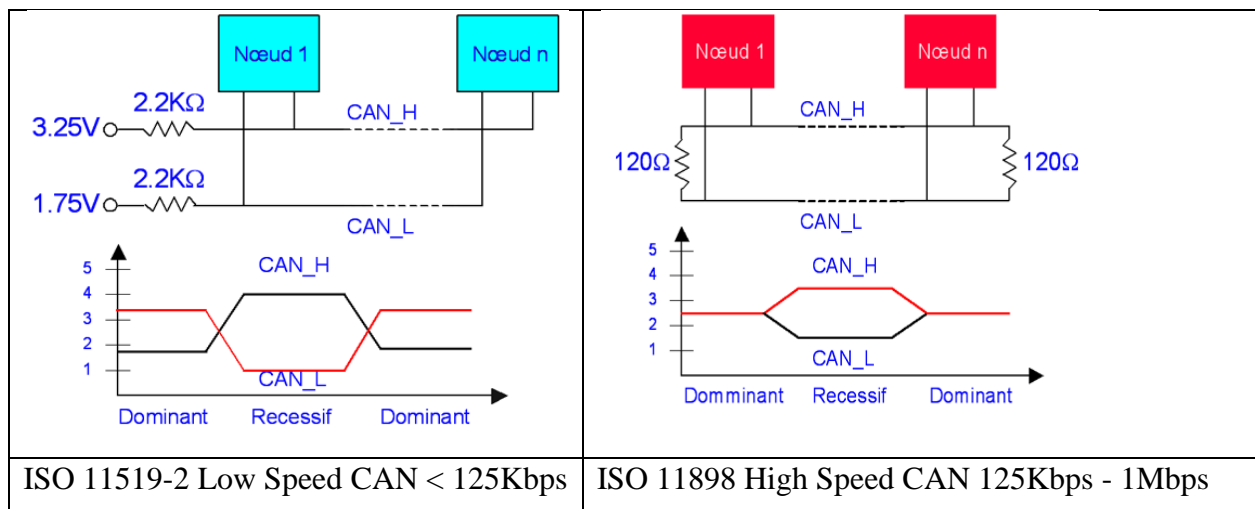


Figure 1.5 : Bus CAN [9].

Les principales différences entre les deux types de bus sont résumées dans le tableau ci-dessous (**Tableau 1.1**).

Tableau 1.1: Comparison entre CAN Low-speed et CAN High-speed [9].

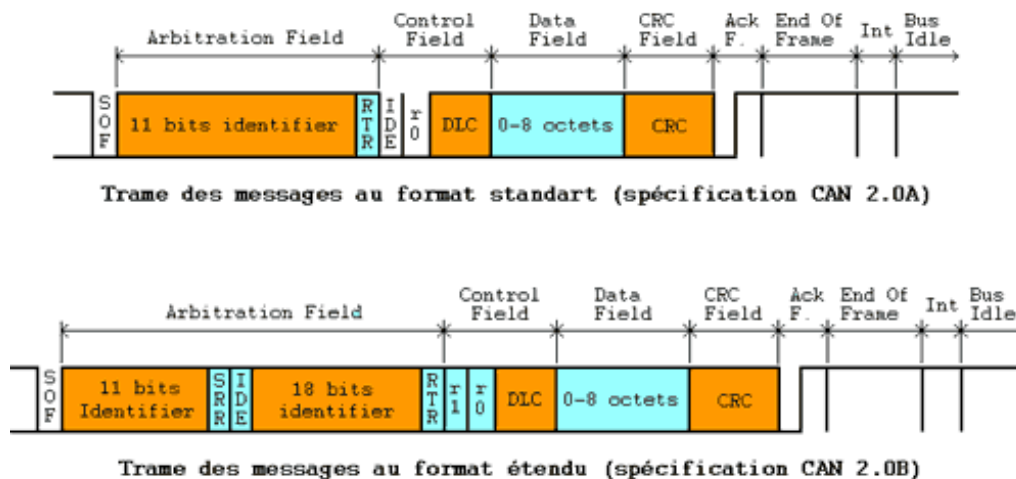
Paramètre	CAN Low-speed	CAN High-speed
Débit	125 Kbit/s	125 Kbit/s à 1 M bit/s
Nombre des nœuds	2 à 20	2 à 30
Résistance de polarisation	2,2kΩ	60 Ω
Courant de sortie en mode émission	1m A	25 à 50 m A
Niveau dominant	CAN H =4V CAN L = 1V	CAN H - CAN L = 2V
Niveau récessive	CAN H =1.75V CAN L = 3.25V	CAN H - CAN L = 2.5V
Caractéristique de câble	30Pf entre les câbles de linge	2 × 120 Ω
Tension d'alimentation	5 V	5 V

Notons que la longueur maximale du câble est de 40 mètre véhiculant un débit de 1 Mbits/s.

Les données se transférant dans la trame admettent deux formats. La différence réside dans le champ « longueur de l'identificateur » (**Figure 1.6**):

Identificateur à 11 bits: est le format CAN standard, appelé CAN 2.0 A.

Identificateur à 29 bits: est le format CAN étendu, appelé CAN 2.0 B.

**Figure 1.6 :** Formats de trame de message du bus CAN [11].

On utilise le protocole CAN dans des applications non-industrielles comme les laboratoires et port automatique ainsi que dans les applications industrielles comme l'escalier mécanique.

Exemple d'application : Il est utilisé dans l'automobile pour faire dialoguer les divers systèmes de contrôle ou de commande : ABS, système de freinage, de suspension, etc. [12]

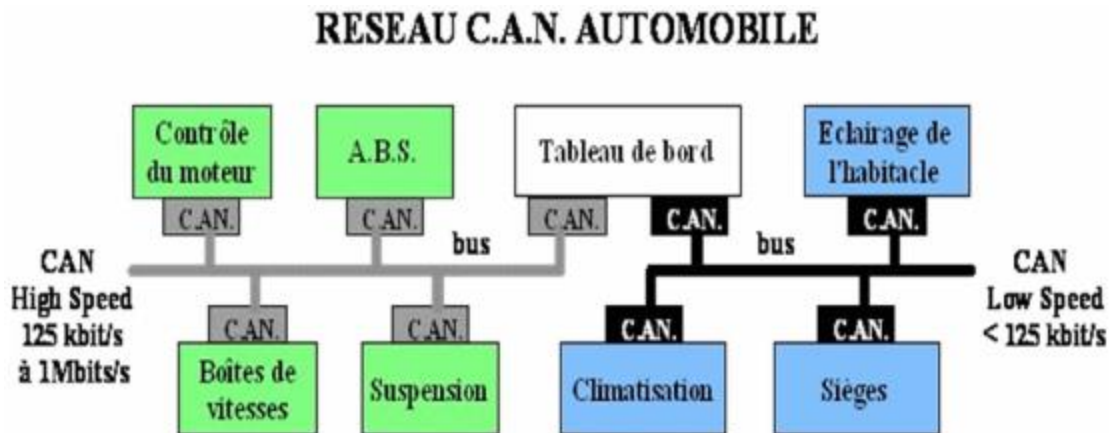


Figure 1.7: Le réseau CAN automobile [12].

1.2.5 I2C (INTER INTEGRATED CIRCUIT)

C'est un protocole de communication série synchrone le plus connu utilisé dans les techniques de processus pour contrôler leurs périphérique.

Il s'agit d'un bus série à deux lignes, l'une pour les données et l'autre pour l'horloge, polarisée par une résistance pull-up dont les valeurs sont comprises entre 2 et 10 K Ω avec une charge capacitive 400pF. La **Figure 1.8** représente un exemple de pochage du bus I2C avec de différents systèmes.

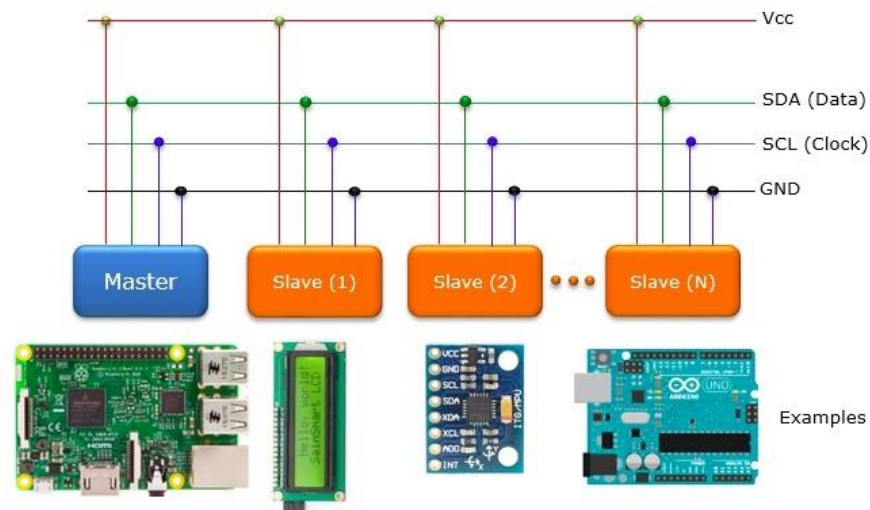


Figure 1.8: bus I2C [13].

Le lecteur est sollicité à lire le Chapitre 2 pour plus de détails.

1.2.6 LIN (Local interconnect network)

Le LIN est un bus de communication série synchrone présent dans des systèmes distribués pour application automobile. Il contient un seul maitre et plusieurs esclaves.

La **Figure 1.9** présente le brochage des nœuds avec le bus LIN. Ces nœuds se composent typiquement d'un microcontrôleur pour gérer le protocole LIN et d'un émetteur-récepteur LIN pour connecter la partie numérique et la ligne physique [14].

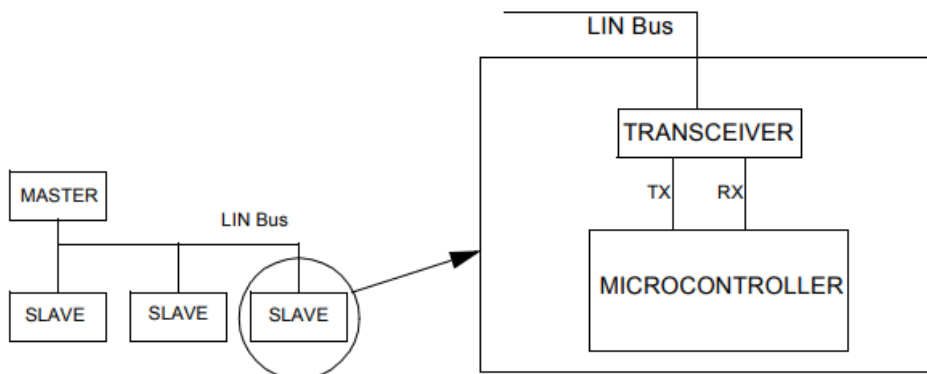


Figure 1.9 : Topologie du bus LIN [14].

Les principales caractéristiques du bus LIN sont : Adresse sur 9 bit. Débits : max 20 kbit/s, Résistance pull-up : 30KΩ au slave, 1KΩ au maître

La **Figure 1.10** montre les bits circulés entre les nœuds.

Sync break field (14 Bits)	Sync field (10bits)	Protected identifier filed (10bits)	DATA 1	DATA 2	DATA N	Checksum (10bits)
			(10- 80bits)			
Message Header			Message Réponse			

Figure 1.10: Trame de bus LIN [15].

Exemple : Il peut être utilisé comme un sous-réseau du bus CAN pour gérer des capteurs comme montré sur la **Figure 1.11**.

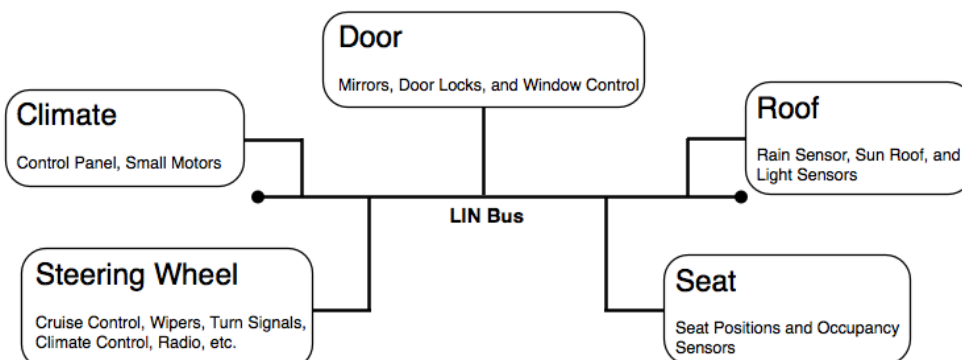


Figure 1.11 : Exemple d'usage du protocole LIN [16].

I.2.7 DVI (Digital Visuel Interface)

C'est une interface visuelle numérique qui a été introduite pour la première fois en 1999 par le groupe de travail sur l'affichage numérique (DDWG). Parmi les avantages du câble DVI on cite [17]:

- 1- Créer une interface informatique standard que tous les écrans et cartes graphiques pourraient partager afin de rationaliser le processus d'installation des ordinateurs, tant pour les entreprises que pour les consommateurs,
- 2- Un câble DVI maximise la qualité des écrans plats LCD.

Le DVI a trois catégories :

- DVI-A : Analogique uniquement (contient 17 pins),
- DVI-D : Numérique uniquement
Single Link (Contient 23 pins)
Dual Link (Contient 25 pins)
- DVI-I : Analogique et numérique
Single Link (Contient 23 pins)
Dual Link (Contient 29 pins).

La **Figure 1.12** présente les différents types de connecteurs selon les catégories.

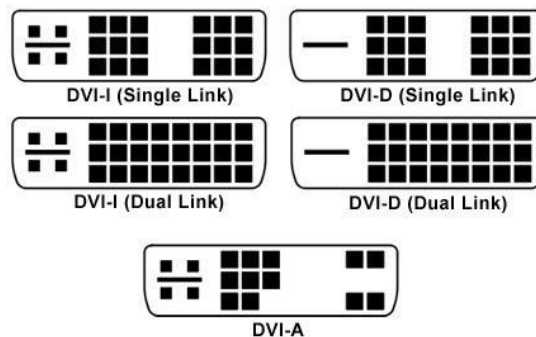


Figure 1.12 : Types de connecteurs DVI femelles [17].

La longueur maximale des câbles DVI n'est pas incluse dans les spécifications car elle dépend de l'horloge des pixels fréquence, et donc les exigences de bande passante du mode vidéo [18] :

- La longueur du câble est d'environ 4.5 m pour la résolution 1920×1080 avec une fréquence de 144 Hz. La bande passante maximale est de 7.44 Gbps.
- La longueur maximale du câble est 15 m pour la résolution d'affichage 1280×1024 avec fréquence d'oscillation 60 Hz.

La **Figure 1.13** montre que la trame DVI est envoyée à un appareil pixel par pixel et ligne par ligne. Chaque pixel se constitue de trois composants : rouge, vert et bleu. L'horloge, quant à elle, sert à la synchronisation [19].

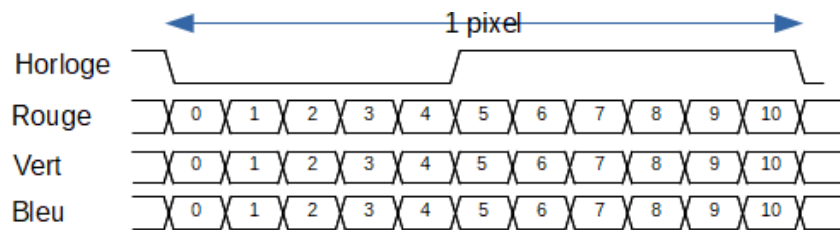


Figure 1.13: Trame de donnée du protocole DVI avec FPGA [19].

1.2.8 HDMI (High-Definition Multimedia Interface)

C'est une norme d'interface utilisée pour l'équipement audiovisuel tel que la télévision haute définition et les systèmes de cinéma maison. Avec 19 fils enveloppés dans un seul câble qui ressemble à un fil USB, HDMI est capable de transporter une bande passante de 5Gbps. C'est plus du double de la bande passante nécessaire pour transmettre l'audio et les vidéo multicanaux [20].

HDMI est capable de transférer de la vidéo, de l'audio et des données non compressés à l'aide d'un seul câble. Les taux de pixels vidéo sont généralement de 25 MHz à 266 MHz (et la vidéo 3D) [21]. La **Figure 1.14** présente différents types de connecteurs HDMI.

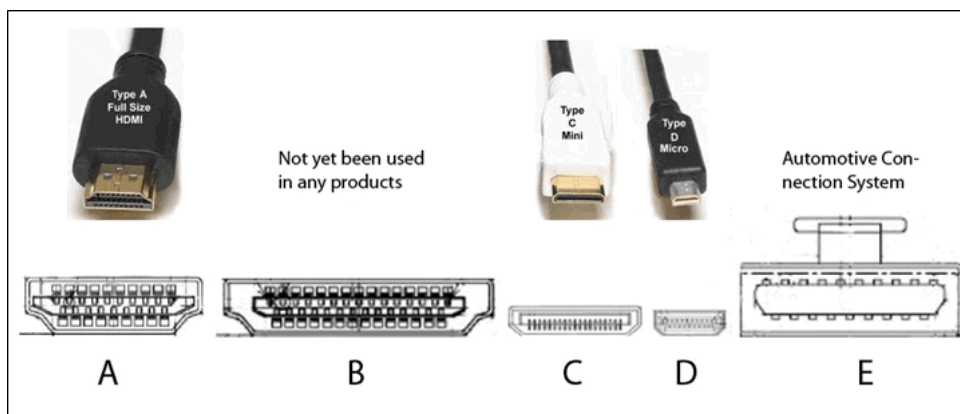


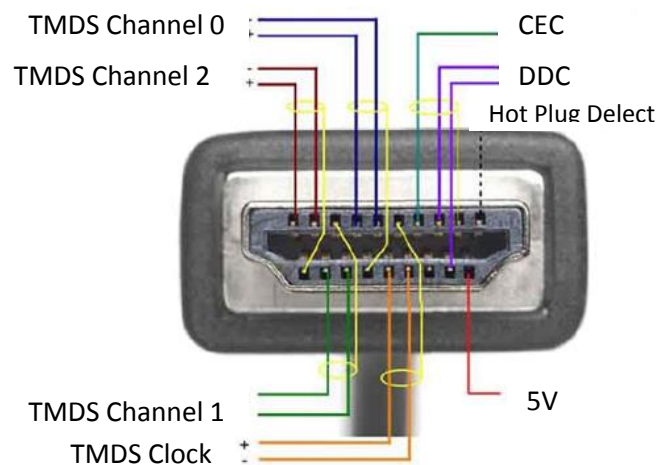
Figure 1.14 : Différents types de connecteurs HDMI [22].

Le tableau ci-dessous (**Tableau 1.2**) donne un aperçu compact sur les méthodes de connexion courantes du protocole HDMI.

Tableau 1.2 : Comparaisons entre les ports HDMI [23].

	HDMI 1.4	HDMI 2.0	HDMI 2.1
Les résolutions les plus importantes	3840 × 2160p -24Hz 1920 × 1080p -24Hz	3840 × 2160p -60Hz 1920 × 1080p -144Hz	7680 × 4320p -60Hz 3840 × 2160p -60Hz (3D)
Bande passante maximale	10.2 Gbit/s	18 Go/s	38,4 Gbits/s
Transmission du son	Oui	Oui	Oui
Transmission 3D	Oui	Oui	Oui
Daisy-chaining (plusieurs moniteurs avec un seul câble)	Oui	Oui	Oui

La **Figure 1.15** d'une prise HDMI montre les paires différentielles.

**Figure 1.15** : Prise HDMI [24].

I.2.9 Display-Port

Display-Port est une norme industrielle qui s'adapte à l'adoption croissante de la technologie d'affichage numérique dans les PC industriels. Elle fournit une prise en charge des applications pour un débit de canal de liaison allant jusqu'à 21.6 Gbit/s. En outre, elle répond aux besoins à long terme de l'industrie des PC pour prendre en charge des définitions d'image supérieures à 2048×1536 et des profondeurs de couleur supérieures à 24 bits. Elle prend en charge la transmission à bande passante réduite via un entraînement direct sur un câble de 15 mètres. Display-Port prend aussi en charge un minimum de lignes 1080p à 24 bpp, avec des

fréquences d'oscillation verticale de 50 et 60Hz sur 4 voies à 15 mètres [25]. Le **Tableau 1.3** reporte une comparaison entre les ports du Display-Port en termes de caractéristiques.

Tableau 1.3 : Comparaisons entre les ports Display Port [23].

	Display Port 1.1	Display Port 1.2	Display Port 1.3	Display Port 1.4
Les résolutions les plus importantes	3840 × 2160p – 30 Hz 1920 × 1080 – 144 Hz	3840 × 2160p – 60 Hz 1920 × 1080 – 144 Hz	7680 × 4320 – 30 Hz 3840 × 2160 – 120 Hz 1920 × 1080 – 144 Hz	7680 × 4320 – 60 Hz 3840 × 2160 – 120 Hz 1920 × 1080 – 144 Hz
Bande passante maximale	8.64 Gbits/s	17,28 Gbits/s	25.9 Go/s	32.4 Go/s
Transmission du son	Oui	Oui	Oui	Oui
Transmission 3D	Non	Oui	Oui	Oui
Daisy-chaining (plusieurs moniteurs avec un seul câble)	non	Oui	Oui	Oui

La **Figure 1.16** présente, quant à elle, le paquet de données Display-Port transmis à débit fixe (choix des débits de liaisons et du nombre de voies d'interface). Les connecteurs de type Display-Port sont affichés à la **Figure 1.17**.

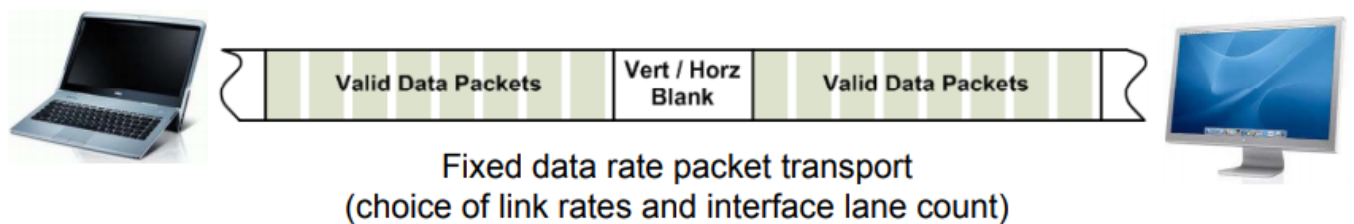
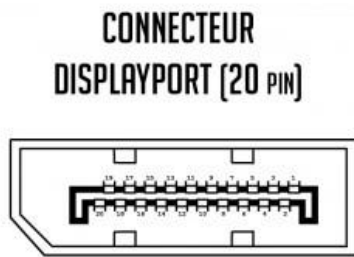


Figure 1.16 : Paquet de donnée Display-Port [26].

Connecteur Display-Port



Connecteur Mini Display-Port

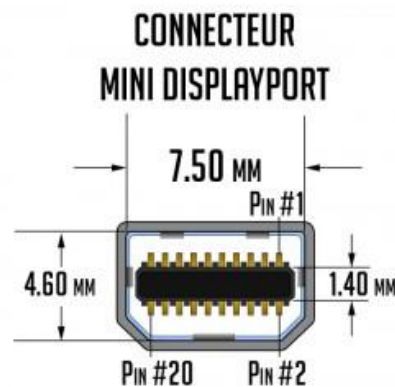


Figure 1.17 : Schéma des connecteurs Display-Port [27].

I.2.10 HART (Highway Addressable Remote Transducer)

C'est un protocole de communication conçu pour applications de mesure et de contrôle de processus industriels. Appelé un protocole hybride parce qu'il combine la communication analogique et numérique. Ça peut communiquer une seule variable à l'aide d'un signal analogique 4-20 mA. L'information numérique est véhiculée par une modulation de bas niveau superposée à la boucle de courant standard de 4 à 20 mA. Il est très approprié pour une utilisation sur le terrain pour économiser le câble de connexion [28]. Les appareils HART peuvent fonctionner dans l'une des deux configurations de réseau : point à point ou multipoint illustrées à la Figure 1.18.

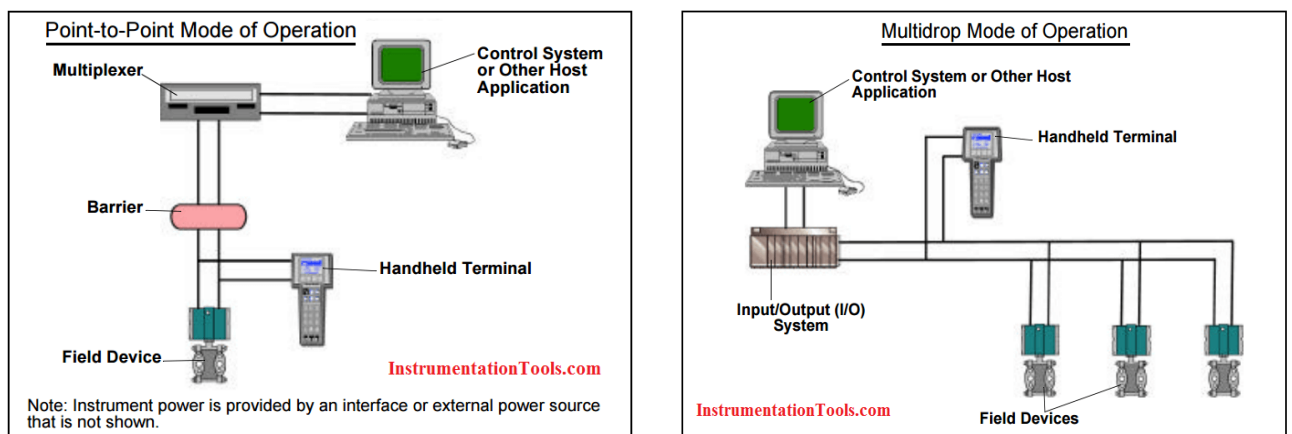


Figure 1.18 : Mode multipoint et point à point configuration réseau [29].

Le format de trame de message HART, souvent appelé télégramme HART, est illustré dans la Figure 1.19. Il se compose de neuf champs.

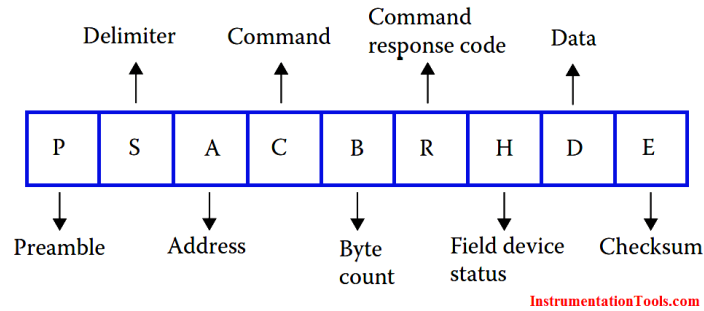


Figure 1.19: Trame de message HART [30].

I.2.11 UART (Universal Asynchronous Receiver/Transmitter)

L'émetteur- récepteur asynchrone universel est un protocole de communication matériel qui utilise une communication série asynchrone avec une vitesse configurable. Asynchrone signifie qu'il n'y a pas de signal d'horloge pour synchroniser les bits de sortie du dispositif de transmission allant à l'extrémité de réception [31].

Les désignations RS-232, RS-422 et RS-485 font référence aux interfaces pour la transmission de données numériques. Les interfaces RS-422 et RS-485 sont largement utilisées dans l'industrie pour connecter divers équipements. Le **Tableau 1.4** montre les principales différences entre les interfaces RS-232, RS-422 et RS-485

Tableau 1.4: Les principales différences entre les interfaces RS-232, RS-422 et RS-485 [32].

Nom de port	RS-232	RS-422	RS-485
Type de transfert	Full-Duplex	Full-Duplex	Half-duplex (2 fils) Full-duplex (4 fils)
Distance maximale	15 mètre à 9600bps	1200 mètre à 9600bps	1200 mètre à 9600bps
Contact in use	TxD, RxD, RTS, CTS, DTR ,DSR, DCD, GND	TxA, TxB, RxA, RxB, GND	Data A, Data B, GND
Topologie	Point à point	Point à point	Multipoint
Nombre d'appareils connectés maximal	1	1(10 appareils en mode réception)	32 (avec un nombre de répéteurs plus élevé généralement jusqu'à 256)

Le mode de transmission est sous forme de paquet. La pièce qui relie l'émetteur et le récepteur inclut la création de paquets série et contrôle ces lignes matérielles physiques. Un

paquet se compose d'un bit de départ, d'une trame de données, d'un bit de parité et de bits d'arrêt. Illustrée à la **Figure 1.20** [31].

Start Bit (1 bit)	Data frame (5to 9 data Bits)	Parity Bits (0 to 1bit)	Stop Bits (2bits 1to)
----------------------	---------------------------------	----------------------------	--------------------------

Figure 1.20 : Paquet UART [31].

I.2.12 PCI (Peripheral Component Interconnect) et PCI E (PCI Express)

Le bus PCI est un bus synchrone supportant un multiplexage de signaux d'adressages et de données. Il est avant tout prévu pour travailler avec des systèmes 32 bits, dans un fonctionnement en échange entre un maître et un esclave sous le contrôle d'un arbitre. [32]. La **Figure 1.21** représente l'architecture classique d'utilisation du bus PCI,

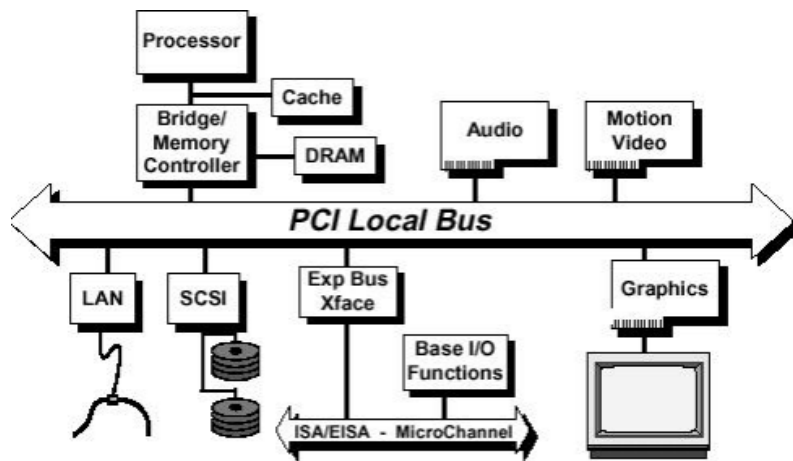


Figure 1.21 : Schéma bloc d'une architecture PCI [32].

Le **Tableau 1.5** présente les caractéristiques paramètres de bus PCI, qui il existe deux paramètre essentiel caractérisant ces bus est la largeur du bus et la vitesse d'horloge :

Tableau 1.5 : Caractéristiques du bus PCI [33].

Variante PCI	Vitesse d'horloge	Largeur du bus		Débit de données de pointe	Débit de données pratique
		bits	bytes		
PCI conventionnel uniquement	33 MHz	32 bits	4 bytes	132 MB/s	90 MB/s
		64 bits	8 bytes	264 MB/s	180 MB/s
PCI conventionnel ou PCI-X	66 MHz	32 bits	4 bytes	264 MB/s	180 MB/s
		64 bits	8 bytes	528 MB/s	360 MB/s
PCI-X uniquement	133 MHz	32 bits	4 bytes	532 MB/s	360 MB/s
		64 bits	8 bytes	1064 MB/s	720 MB/s

Le transfert de données PCI peut être accompli à l'aide de "transferts burst" (voir la **Figure 1.22**). L'adresse et le type de transfert sont émis pendant la phase d'adressage, Un objet

de données (jusqu'à 32 bits (dans une implémentation 32 bits) ou 64 bits (dans une implémentation 64 bits) peut alors être transféré au cours de chaque phase de données subséquente [34]



Figure 1.22 : Exemple de transfert du burst data [34].

Le **PCI Express (Peripheral Component Interconnect Express)** a remplacé l'ancienne norme PCI. Il s'agit d'une série standard de bus d'extension pour connecter un ordinateur à un ou plusieurs dispositifs périphériques [35].

Le bus PCI Express se décline en plusieurs versions, 1X, 2X, 4X, 8X, 12X, 16X et 32X, permettant d'obtenir des débits compris entre 250 Mo/s et 8 Go/s. [36]. La **Figure 1.23** présente les quatre emplacements standards PCI express.

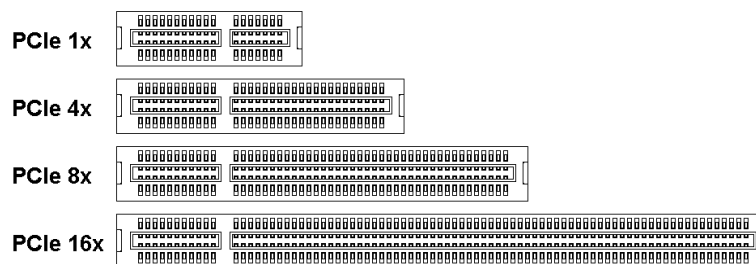


Figure 1.23 : Quatre emplacements du PCIe standard [37].

I.2.13 USB (Universal serial bus) [38]

L'USB est une interface qui relie un appareil avec un ordinateur pour l'envoi et la réception de données. La résistance pull-up (valeur de résistance est $1.5K\Omega$) est placée sur la ligne D+ ou D- pour détecter la vitesse de l'appareil. La distance maximale du câble est de 25 mètres avec un débit maximal de 480 Mbits/s.

Si la ligne est D+ : Indique que le périphérique connecté est un périphérique à vitesse maximum.
Si la ligne est D- : Indique que le périphérique connecté est un périphérique à faible vitesse, comme représenté sur la **Figure I.24**. Le paquet de données est rapporté sur la **Figure 1.25**.

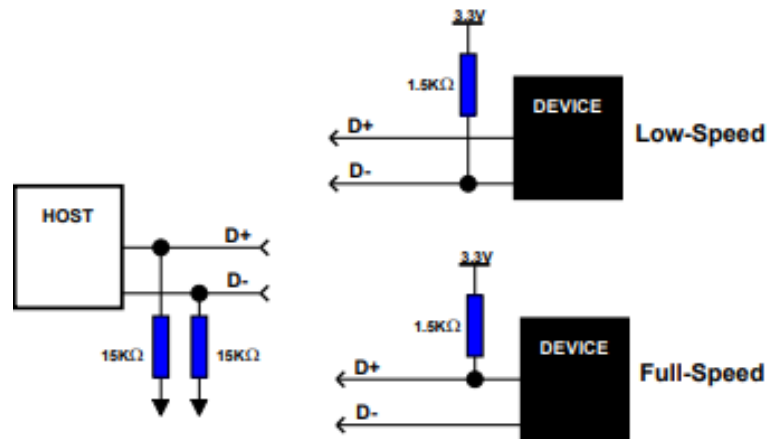


Figure 1.24 : Détection de vitesse USB [38].

Packet	Dir	F	Sync	SOF	Frame	CRC5	EOP
355	→	S	00000001	0xA5	951	0x13	250.000 NS

Figure 1.25 : La trame USB [39].

Il existe 3 normes d'USB spécifiant la vitesse de transmission maximales, comme illustrées dans le **Tableau 1.6** [40].

Tableau 1.6 : Vitesses de transmission maximales selon les différentes normes USB [40].

Norme USB	Année	Vitesse de transmission maximale
USB 1.1	1998	12 Mbit/s
USB 2.0	2000	480 Mbit /s
USB 3.0	2009	4.8 Gbit/S (soit 600Mbits/s)

1.2.14 CSI et DSI [41]

Avec l'inclusion d'écrans et de caméras haute résolution avec la possibilité de capturer ou de lire des vidéos haute définition, la quantité de transfert de données nécessaire pour atteindre cette haute définition a considérablement augmenté. L'interface (CSI-2) et l'interface série d'affichage (DSI) sont les deux protocoles de haut niveau basés sur des paquets qui transportent des données d'image entre le périphérique et le processeur d'application. Tous les deux utilisent la couche physique D-PHY. La **Figure 1.26** présente le schéma fonctionnel d'un appareil mobile typique.

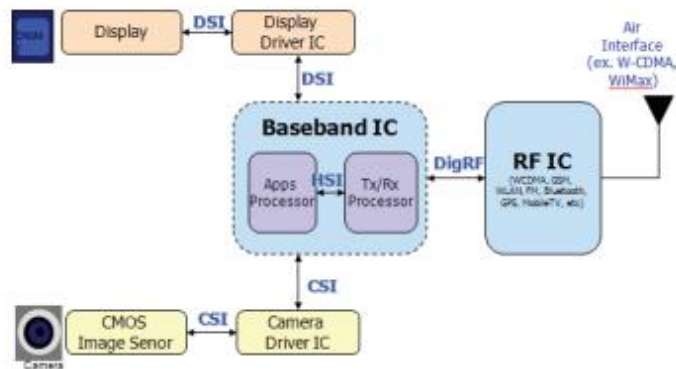


Figure 1.26 : Schéma fonctionnel d'un appareil mobile typique [41].

1.2.14.1 CSI (Camera Serial Interface)[42]

C'est un bus série camera standard, défini par l'alliance MIPI (Mobile Industry Processor Interface). Il définit les bus reliant les différents composants embarqués dans les appareils mobiles comme l'illustrée la **Figure 1.27**.

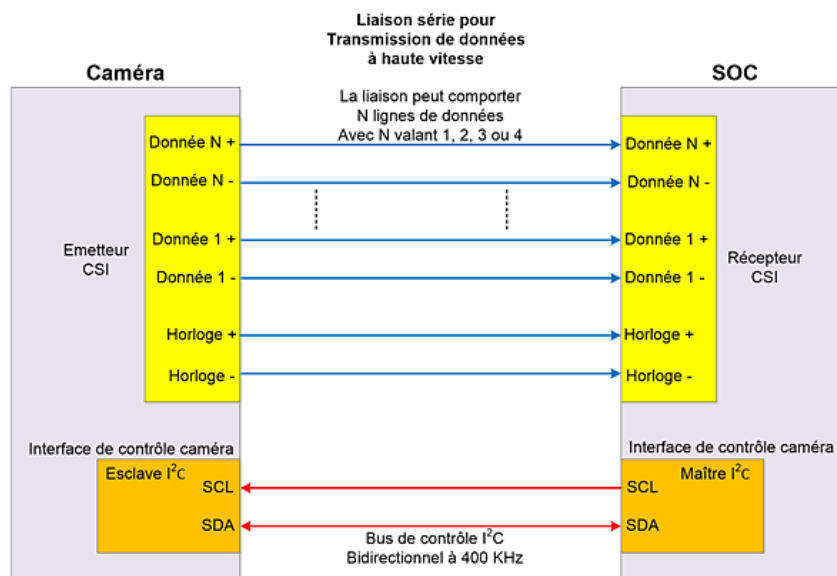


Figure 1.27 : Bus CSI du Raspberry PI [42].

1.2.14.2 DSI (Display Serial Interface) [42]

C'est une spécification de l'alliance MIPI. Son objectif est de réduire le prix des écrans utilisés sur les mobiles. Sa structure est proche de celle du bus CSI, à l'exception près que les lignes de données partent du SoC et vont vers l'écran. La **Figure 1.28** suivante présente l'interface MIPI-DSI tandis qu'un exemple d'utilisation de l'interface DSI est présenté dans la **Figure 1.29**.

1.2.15 SPI (Serial Peripheral Interface) [45]

Le SPI est un port d'entrée/sortie série synchrone à grande vitesse qui permet un bit série flux de longueur programmée (2 à 16 bits) à décaler entrée et sortie de l'appareil à un taux de transfert de bits programmé. Il est généralement utilisé pour communiquer entre l'appareil et les périphériques externes. Les applications typiques incluent une interface externe E/S périphérique ou extension via des périphériques tels que des registres de commutation, des pilotes d'affichage et SPI EPROMs et convertisseurs analogique-numérique.

Le bus d'interface périphérique série (SPI) : La **Figure 1.31** montre comment le périphérique esclave est connecté au maître dans l'implémentation SPI maître unique, esclave unique et la **Figure 1.32** montre les implémentations SPI à maître unique et à esclaves multiples.

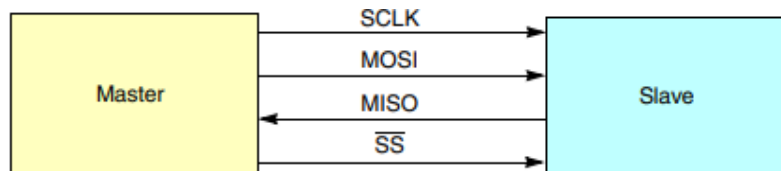


Figure 1.31: Implémentation SPI maître unique, esclave unique [46].

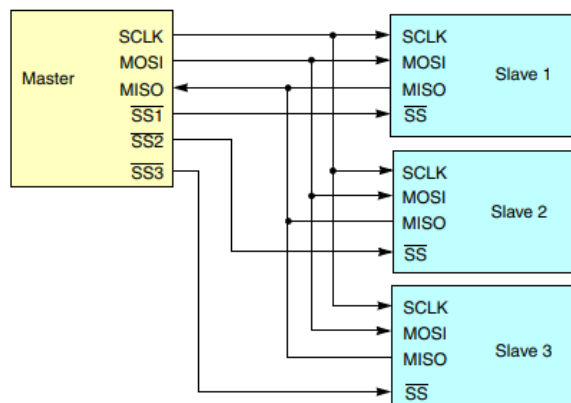


Figure 1.32: Mises en œuvre de plusieurs esclaves SPI [46].

Pour connecter le maître et l'esclave on a quatre lignes de liaison. Le **Tableau 1.7** présente les lignes externes de ce bus.

Tableau 1.7 : Description des signaux [46].

Signal	Description
SCLK	Signal d'horloge série avec la direction de l'appareil maître à esclave.
MOSI	Signal master-out/slave-in pour la transmission de données de maître à esclave en 8 bits, 12 bits, 16 bits, Largeur 20 bits, 24 bits ou 32 bits (sortie du maître).
MISO	Signal d'entrée/de sortie d'esclave pour la transmission de données d'esclave à maître en 8 bits, 12 bits, 16 bits, Largeur 20 bits, 24 bits ou 32 bits (sortie de l'esclave).
SS	Slave select signal, initiated by master to select slave device.

1.2.16 SAS (Serial Attached SCSI), SATA (Serial Advanced Technology Attachment)

1.2.16.1 SATA (Serial Advanced Technology Attachment) [47]

Est une interface série full-duplex transmettre de donnée qui permet de connecté une carte mère avec nombreuse périphérique telle que disque dur et lecteur DVD) à technologie point à point qui peut prendre en charge plusieurs appareils mais il nécessite plusieurs linge [48]. **Trame SATA** se composent (Illustrée à la **Figure 1.33**): Un début de trame (SOF) délimité, Informations sur la couche de transport de charge utile, Contrôle de Redondance Cyclique (CRC), Un délimiteur de fin de trame (EOF). [47]

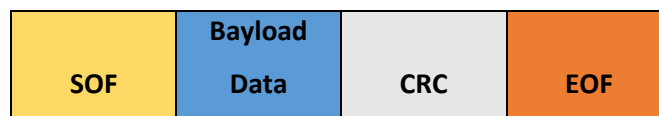


Figure 1.33 : Trame de donnée SATA [47].

La **Figure 1.34** présente les câbles à norme SATA.

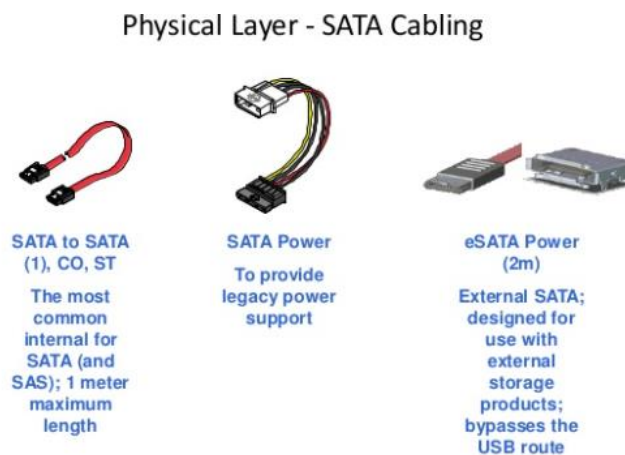


Figure 1.34: Câbles SATA [47].

1.2.16.2 SAS (Serial Attached SCSI)

C'est une interface matérielle standard pour les lecteurs de stockage. SAS est une architecture point à point qui utilise un contrôleur (adaptateur de bus hôte) avec deux canaux full-duplex qui fonctionnent simultanément. Chaque canal (port SAS) transfère les données à 3, 6 ou 12 Gbit/s dans chaque direction [48].

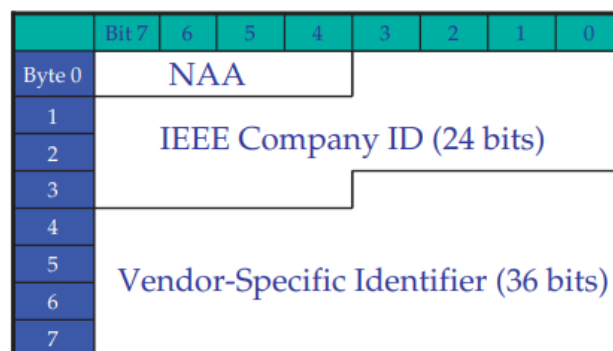
Les canaux SAS (ports) peuvent se connecter à un seul lecteur ou à un module d'extension qui connecte jusqu'à 128 lecteurs (ou adaptateurs de bus hôte). Un Fanout Expander peut connecter des Edge Expanders et un système peut prendre en charge 16 256 disques [48]. Le **Tableau 1.8** répertorie les débits de liaison physique pour SAS et SATA.

Tableau 1.8 : Taux de liaison physique par direction [49].

Génération	Taux de liaison physique	Bande passante pour 1 connexion	4×Bande passante
SAS-1,SAS-1,1,SATA Revision 1.0	1.5 Gbits/s	150 MB/s	600 MB/s
SAS-1,SAS-1,1,SATA Revision 2.0	3 Gbits/s	300MB/s	1200MB/s
SAS-2,SAS-2,1,SATA Revision 3.0	6 Gbits/s	600 MB/s	2400MB/s

Les adresses utilisées dans SAS sont des unités 64 bits qui se composent de trois parties, comme illustrées à la **Figure 1.35** [50] :

- Identifiant de 4 bits attribué par l'autorité d'adresse de nommage. Cette valeur est la même pour SAS et Fibre Channel, qui est de 5,
- Un identifiant d'entreprise 24 bits attribué par l'IEEE,
- Un identifiant spécifique au fournisseur de 36 bits.

**Figure 1.35** : Adresse SAS [50].

Comparaison entre SAS et SATA :

Les disques SAS utilisent le même facteur de forme que Serial ATA (SATA) pour le connecteur de câble, mais ajoutent un deuxième port pour la connexion à deux contrôleurs ou extensions pour la tolérance aux pannes.

1.2.17 IO-LINK :

IO-Link est une interface de communication point à point innovatrice pour le domaine des capteurs/actionneurs, spécifiée dans la norme CEI 61131-9[51]. La **Figure 1.36** montre la trame de donnée IO-Link et la séquence message.

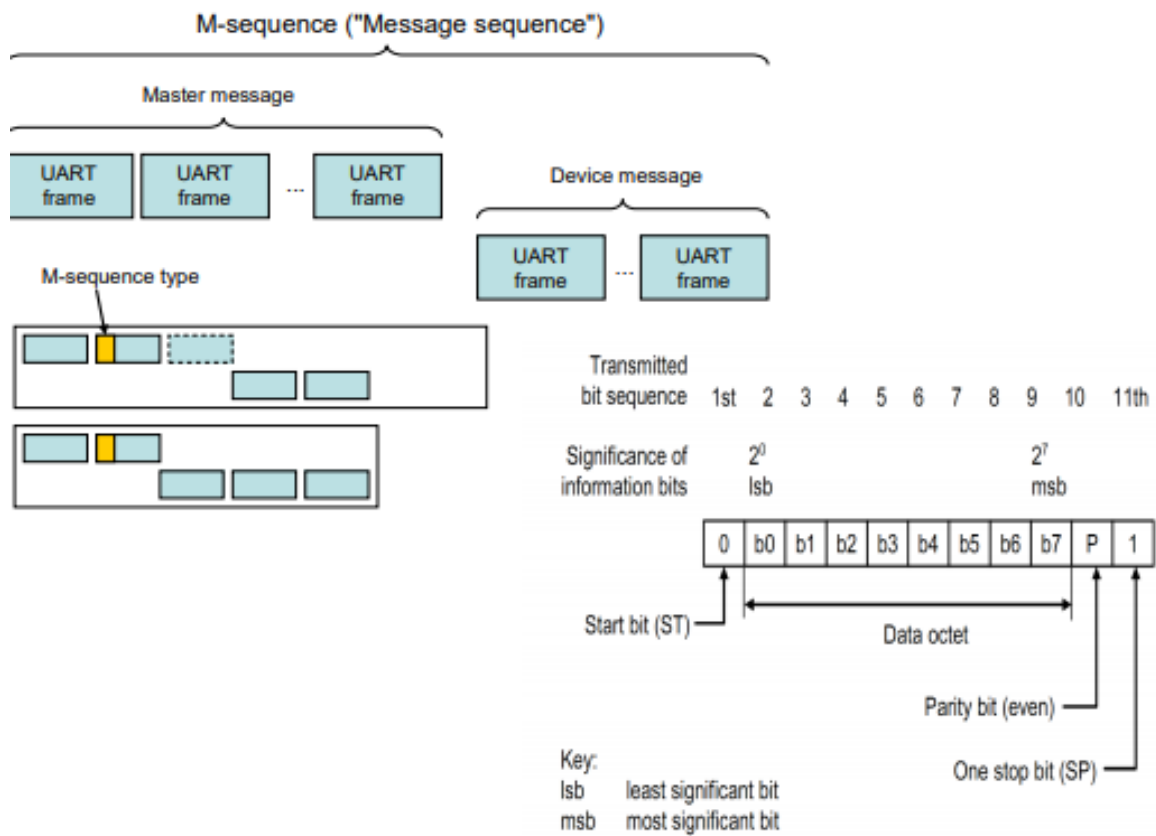


Figure 1.36 : Séquence de communication maître- Device IO-Link [52].

1.2.18 Ethernet

C'est un protocole de communication série, de type topologies bus ou étoile, On utilise un câble pour relier les équipements informatique entre eux soit câble cuivre ou câble fibre optique. La longueur di câble varie entre 500 m à 20 km, avec un débit maximale de 40Gbits/s [53].

La **Figure I.37** présente le pochage de plusieurs équipements par le bus Ethernet tandis que la **Figure I.38** montre les bits circulés entre les terminaux.

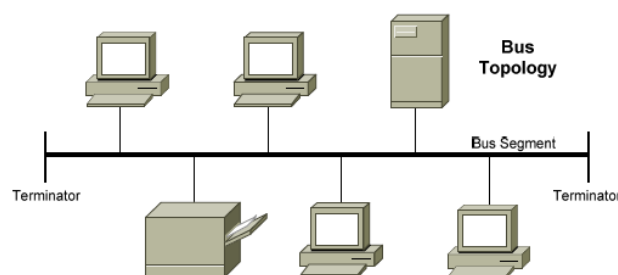


Figure 1.37: Bus Ethernet [54].

Preamble	SFD	Destination address	Source address	Type	Data	CRC
7 bytes	1 byte	48 bits	48 bits	48 bits	46-1500 bytes	32 bits

Figure 1.38 : Paquet Ethernet [53].

Le modèle de communication TCP/IP est illustré à la **Figure 1.39**.

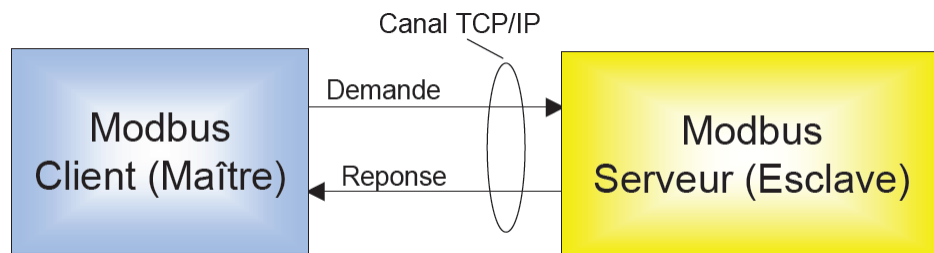


Figure 1.39 : Le modèle de communication du TCP/IP.

1.3 Conclusion

Dans ce chapitre, nous avons donné un aperçu général sur les protocoles de communication les plus utilisés en pratique. Dans le prochain chapitre, nous nous focalisons sur le protocole I2C qui sera implémenté sur une carte de développement, la DE1, afin de permettre le contrôle d'un codeur/décodeur audio.

II.1 Introduction

Le bus I2C (Inter Integrated Circuit) a été développé au début des années 80 par Philips semi-conductors pour permettre de relier facilement à un microprocesseur les différents circuits d'un téléviseur moderne. Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils : Un signal de donnée (SDA), un signal d'horloge (SCL), et un signal de référence électrique (Masse). Ceci permet de réaliser des équipements ayant des fonctionnalités très puissantes (en apportant toute la puissance des systèmes micro programmés) et conservant un circuit imprimé très simple, par rapport à un schéma classique (8bits de données, 16 bits d'adresse+ les bits de contrôle).

Les données sont transmises en série à 100Kbits/s en mode standard et jusqu'à 400Kbits/s en mode rapide. Ce qui ouvre la porte de cette technologie à toutes les applications où la vitesse n'est pas primordiale. De nombreux fabricants ayant adopté le système, la variété des circuits disponibles disposant d'un port I2C est énorme : Ports d'E/S bidirectionnels, convertisseurs A/N et N/A, mémoires (RAM, EPROM, EEPROM, etc..), circuits audio (Equaliseur, Contrôle de volume, ...) et autres drivers (LED, LCD, ...). Le nombre de composants qu'il est ainsi possible de relier est essentiellement limité par la charge capacitive des lignes SDA et SCL : 400 pF.

II.2. Le bus de communication I2C

L'objectif est de permettre la communication entre plusieurs équipements. Il existe une très grande variété de bus de communication car aucun bus n'est universel. Il faut choisir le bus en fonction de [55]:

- Distance entre les équipements (cm, m, km),
- Nombre d'équipements à relier,
- Débit de données (contrôle ou data),
- Consommation autorisée (pile/secteur),
- Stabilité nécessaire (bruit),
- Catalogue disponible,
- Coût (en générale, la bonne solution est trop chère).

La structure du bus I2C est illustrée sur la **Figure 2.1**.

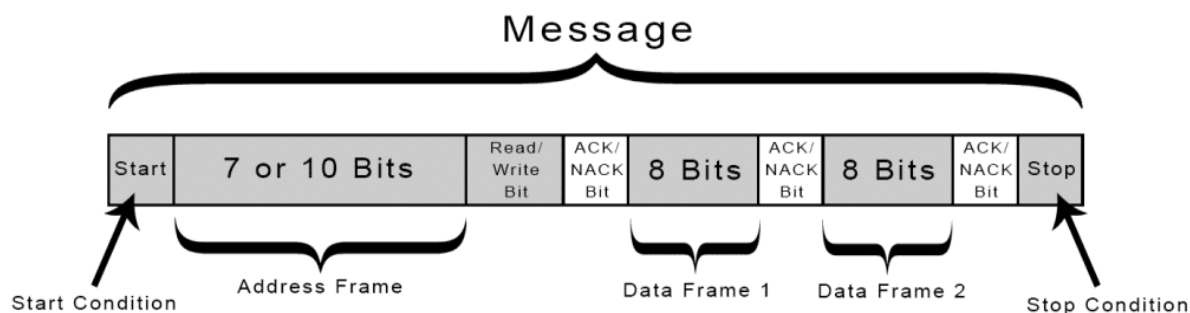


Figure 2.1: Bus I2C [56].

II.2.1 Etude du bus I2C

Le bus I2C permet de faire communiquer entre des composants électroniques grâce à trois fils (**Figure 2.2**) :

- Signal de données SDA,
- Signal d'horloge SCL,
- Signal de référence électrique GND (masse).

Le **Tableau 2.1** reporte les caractéristiques principales du bus I2C.

Tableau 2.1 : Les caractéristiques du bus I2C.

Fils utilisés	2
Vitesse maximum	Mode standard =100Mbps Mode rapide=400Kbps Mode Grande vitesse=3,4Mbps Mode ultra rapide=5Mbps
Synchrone ou asynchrone ?	Synchrone
Série ou parallèle	Série
Max #of Masters	Illimitée
Max #of Slaves	1008

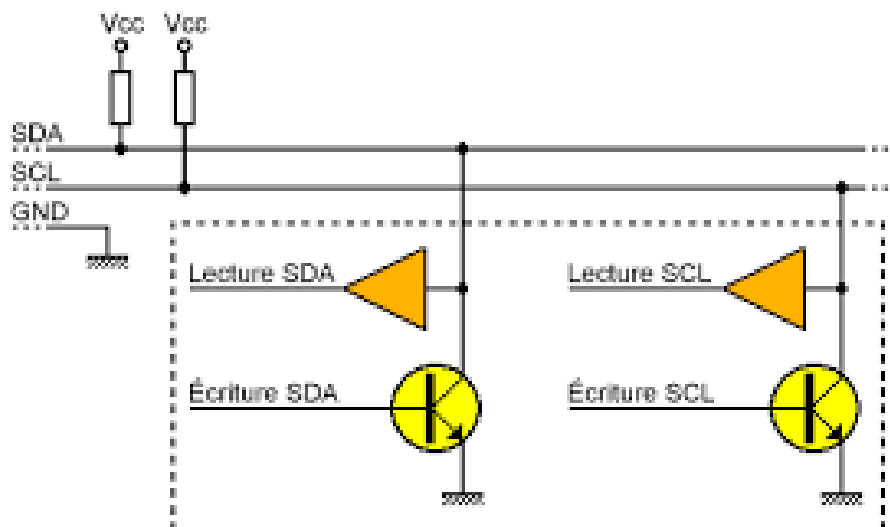


Figure 2.2 : Structure d'E/S d'un module I2C [57].

II.3 Le protocole I2C

Le protocole I2C définit la succession des états logiques possibles sur SDA et SCL et la façon dont doivent réagir les circuits en cas de conflits.

II.3.1 Glossaire I2C

Le **Tableau 2.2** résume le glossaire I2C.

Tableau 2.2 : Glossaire I2C [58].

Abonné	Tout élément connecté sur le bus
Emetteur	Tout abonné qui envoie des données sur SDA
Récepteur	Tout abonné qui reçoit des données de SDA
Maître	Tout abonné qui démarre et termine un échange. Le maître place l'horloge sur SCL
Esclave	Tout abonné adressé par un maître. Un esclave a la possibilité d'arrêter l'horloge du maître.
Adresse	Numéro attribué à un esclave. Sur le bus tous les esclaves ont une adresse unique
Echange	Dialogue entre un maître et un esclave
Arbitrage	Résolution du conflit d'un accès simultané par 2 maîtres.

II.3.2. Câblage I2C

Les lignes SCL et SDA sont par défaut à VDD. Pour mettre 1 sur SCL ou SDA, un abonné programme le port en entrée, la résistance R_p se charge de tirer la ligne à 1. Pour mettre 0 sur SCL ou SDA, un abonné doit écrire un 0, c.-à-d. relier la ligne à la masse. Il ne peut jamais y avoir de conflit électrique (court-circuit VDD-GND) comme le montre la **Figure 2.3**.

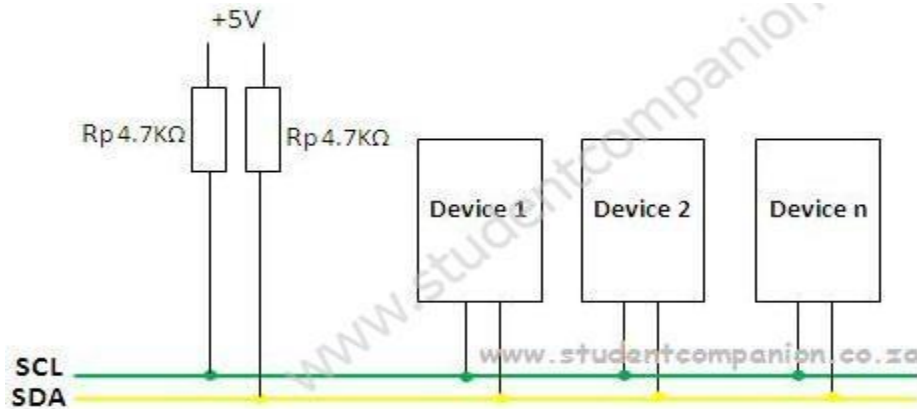


Figure 2.3: Câblage I2C [59].

II.3.3. La prise de contrôle du bus I2C

Pour prendre le contrôle du bus, il faut que celui-ci soit au repos (SDA et SCL à '1'). Pour transmettre des données sur le bus, il faut donc surveiller deux conditions particulières [55]:

- La condition de départ (SDA passe à '0' alors que SCL reste à 1)
- La condition d'arrêt (SDA passe à '1' alors que SCL reste à '1')

Lorsqu'un circuit, après avoir vérifié que le bus est libre, prend le contrôle de celui-ci, il en devient le maître. C'est lui qui génère le signal d'horloge. Un schéma décrivant la condition de départ et d'arrêt est illustré à la **Figure 2.4**.

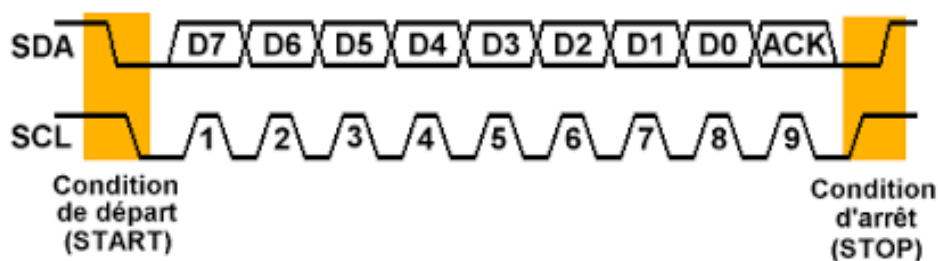


Figure 2.4: Condition de départ et d'arrêt [60].

II.3.4 . Ecriture d'un octet (une donnée) [61]

Avant de placer les bits qui forment l'octet à transmettre sur le bus, le maître doit placer la ligne d'horloge SCL à 0. Tant que la ligne SCL est au niveau haut, la ligne SDA ne doit pas changer d'état, sinon cette condition sera interprétée comme la condition d'arrêt. Celle-ci peut survenir même au milieu de la transmission d'un octet, pour abandonner la transmission et libérer le bus pour les autres circuits. Pour transmettre correctement les bits sur la ligne SDA, le maître doit donc tout d'abord placer la ligne SCL à 0. Ensuite, le maître peut placer la ligne SDA au niveau correspondant au bit à transmettre et replacer la ligne SCL au niveau 1 pour indiquer que le bit est présent sur la ligne SDA. La même opération va se répéter autant de fois que nécessaire pour transmettre les 8 bits de données. Notez que c'est le bit de poids fort qui est transmis en premier.

Une fois les 8 bits transmis, le circuit qui vient de recevoir les données doit imposer un bit d'acquittement ACK sur la ligne SDA. Pour cela, pendant que la ligne SCL est au niveau bas, le maître place sa propre sortie au niveau haut, tandis que le récepteur (aussi appelé l'esclave) place sa sortie au niveau bas. Puisque les sorties sont à collecteur ouvert, la ligne SDA restera au niveau bas à cause de l'esclave. Le maître relit ensuite la ligne SDA une fois qu'il a passé la ligne SCL au niveau haut. Si la valeur lue pour le bit ACK est 0, c'est que l'esclave s'est bien acquitté de l'octet reçu, sinon c'est qu'il y a une erreur et le maître doit générer la condition arrêt. Un exemple d'une transmission réussie est représenté sur la **Figure 2.5**. Les étapes d'écriture d'un octet sont reportées sur la **Figure 2.6**.



Figure 2.5 : Exemple de transmission réussie [62].

Dans cet exemple :

- SCL : Horloge imposée par le maître
- SDAM : Niveaux de SDA imposés par le maître
- SDAE : Niveaux de SDA imposés par l'esclave
- SDAR : Niveaux de SDA réels résultants

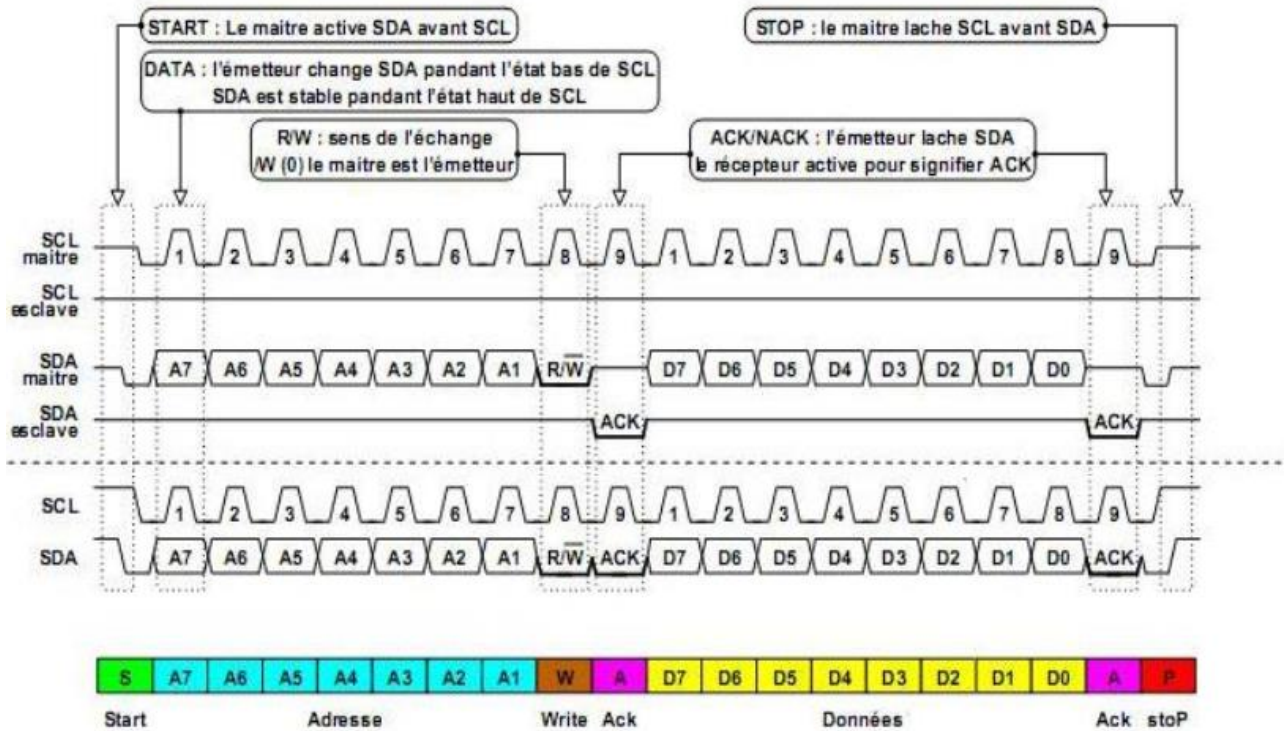


Figure 2.6 : Trame de base : Ecriture d'un octet [63].

II.3.5. Transmission d'une adresse[63]

Le nombre de composants qu'il est possible de connecter sur un bus I2C étant largement supérieur à deux, il est nécessaire de définir pour chacun une adresse unique. L'adresse d'un circuit, codée sur sept bits, est définie d'une part par son type et d'autre part par l'état appliqué à un certain nombre de ces broches. Cette adresse est transmise sous la forme d'un octet au format particulier comme l'explique la **Figure 2.7** suivante.

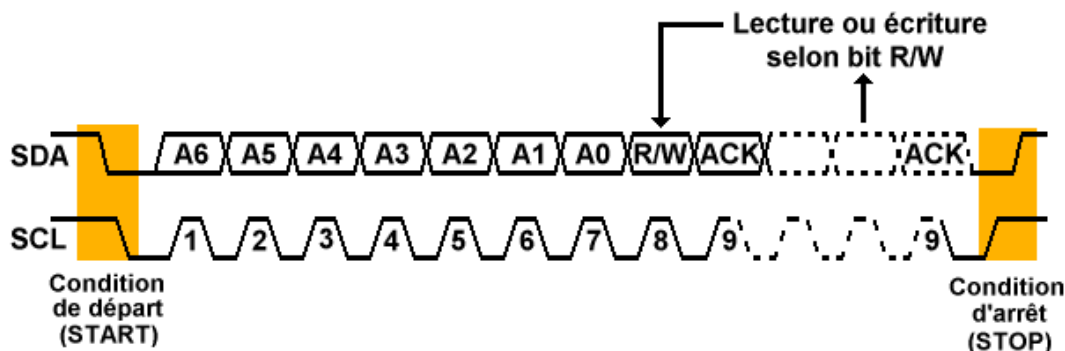


Figure 2.7 : Exemple d'octet d'adresse [64].

On remarque ici que les bits D7 à D1 représentent les adresse A6 à A0, et que le bit D0 est remplacé par le bit de R/W qui permet au maître de signaler s'il veut lire ou écrire une donnée. Le bit d'acquittement ACK fonctionne comme pour une donnée, ceci permet au maître de vérifier si l'esclave est disponible .

Note 1: Cas particuliers des mémoires

L'espace adressable d'un circuit de mémoire étant sensiblement plus grand que la plupart des autres types de circuits, l'adresse d'une information y est codée sur deux octets ou plus. Le premier représente toujours l'adresse du circuit, et les suivants l'adresse interne de la mémoire.

Note 2 : Les adresses réservées

Les adresses 00000XXX et 111111XX sont réservés à des modes de fonctionnement particuliers.

Note 3: Cas particulier d'utilisation d'ACK

L'écriture d'un octet dans certains composants (Mémoires, microcontrôleur.....) peut prendre un certain temps. Il est donc possible que le maître soit obligé d'attendre l'acquittement ACK avant de passer à la suite[63] .

II.3.6. Lecture d'une donnée

La lecture d'une donnée par le maître se caractérise par l'utilisation spéciale qui est faite du bit ACK. Après la lecture d'un octet, le maître positionne ACK à '0' s'il veut lire la donnée suivante (cas d'une mémoire par exemple) ou à '1' (cas échéant). Il envoie alors la condition d'arrêt[12] . La **Figure 2.8** illustre un exemple de lecture d'une donnée tandis que la **Figure 2.9** décrit la trame de base en état d'attente.

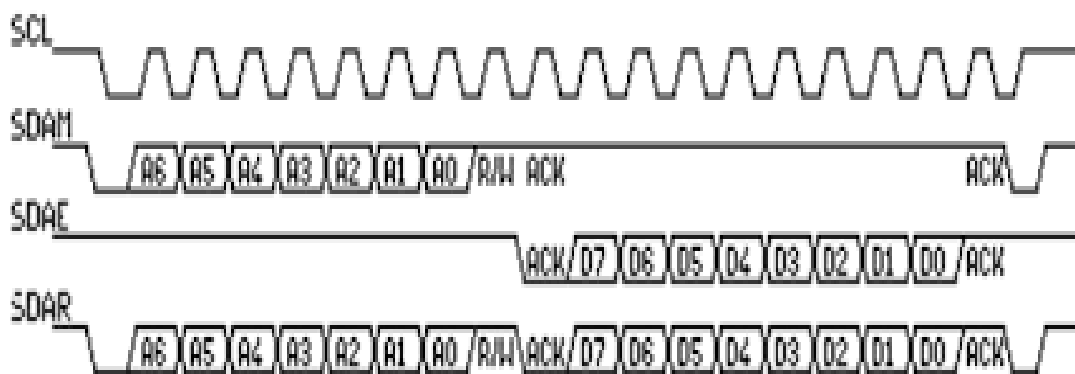


Figure 2.8 : Exemple de lecture d’une donnée [65].

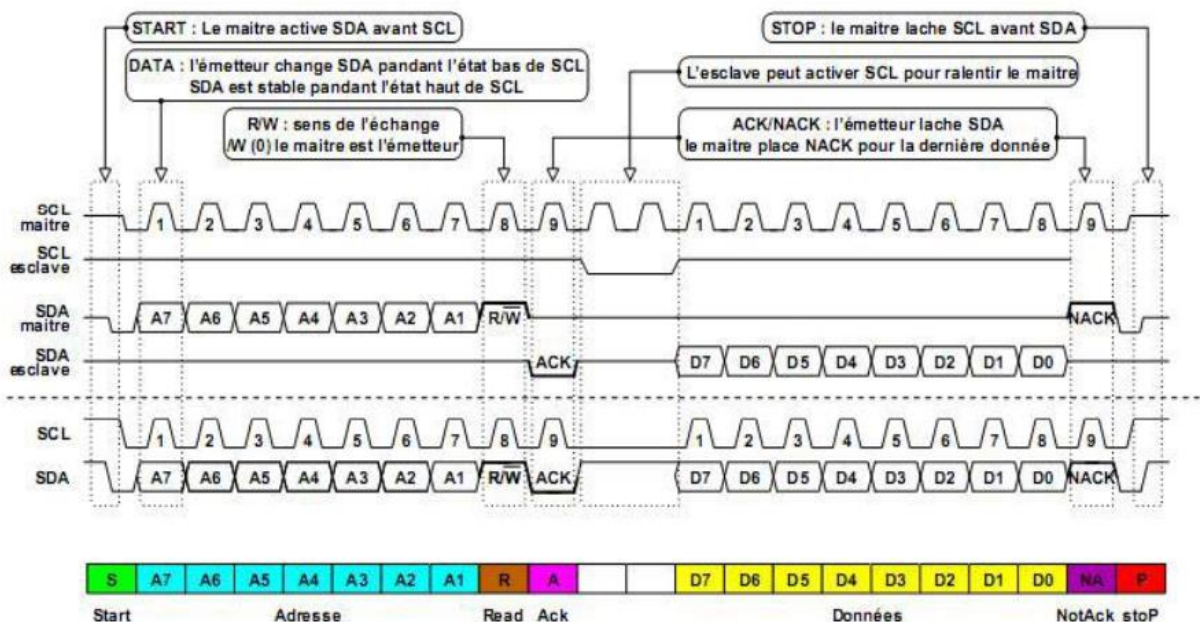


Figure 2.9 : Trame de base : Etat d'attente [66].

II.4 Utilisation de l’I2C

Le protocole I2C est utilisé pour établir la communication entre deux ou plusieurs circuits intégrés, d’où la raison pour la quelle il est connu sous le nom de communication à circuits intégrés (I2C).

II.5 Conclusion

Le protocole I2C est un système de communication série simple et rapide à mettre en œuvre. Il nous permet de construire des bus avec plusieurs maîtres et esclaves.

Ce protocole permet la résolution automatisée des conflits de composants. Ceci est géré par les composants eux-mêmes. De plus, il est pratiquement intéressant du fait que : seuls deux fils sont utilisés, plusieurs maîtres et esclaves sont pris en charge et le bit ACK/NACK vérifie que chaque trame a été transmise avec succès. Le protocole, en revanche, limite la trame de données à 8 bits seulement. De plus, il ne permet aucune vérification des données envoyées. Il est difficile de dire si les données reçues sont correctes dans ce cas.

III.1 Introduction

Comme annoncé dans l'introduction générale, l'objectif principal de ce présent travail est d'implémenter un protocole de communication série, en occurrence, le protocole I2C dans une carte FPGA. Cette implémentation a pour première application pratique le contrôle du signal son.

Pour se faire, nous utilisons la carte de développement DE1 incluant la FPGA **ALTERA Cyclone II**. Nous détaillerons au cours de ce chapitre la réalisation du circuit électronique du protocole **I2C** sur cette carte FPGA programmé avec le langage VHDL de **Quartus ALTERA V 13.0**. Le cahier de charge, pour rappel, consiste à permettre le contrôle de plusieurs paramètres du codec audio **WM8731** à savoir : Augmenter/diminuer le volume, muter/écouter et enfin démarrer/arrêter.

III.2 Codec audio et communication I2C

III.2.1 Schéma fonctionnel du codec audio WM8731

Le WM8731 est un circuit intégré que l'on trouve dans la carte de développement DE1 qui permet de coder et décoder l'audio et le flux de données numériques. Il contient des convertisseurs : analogique-numérique (ADC) et numérique-analogique (DAC). Ce composant comporte un ensemble des registres de programmations qui se réalisent grâce à l'I2C. La **Figure 3.1** illustre le schéma fonctionnelle du codec audio WM8731.

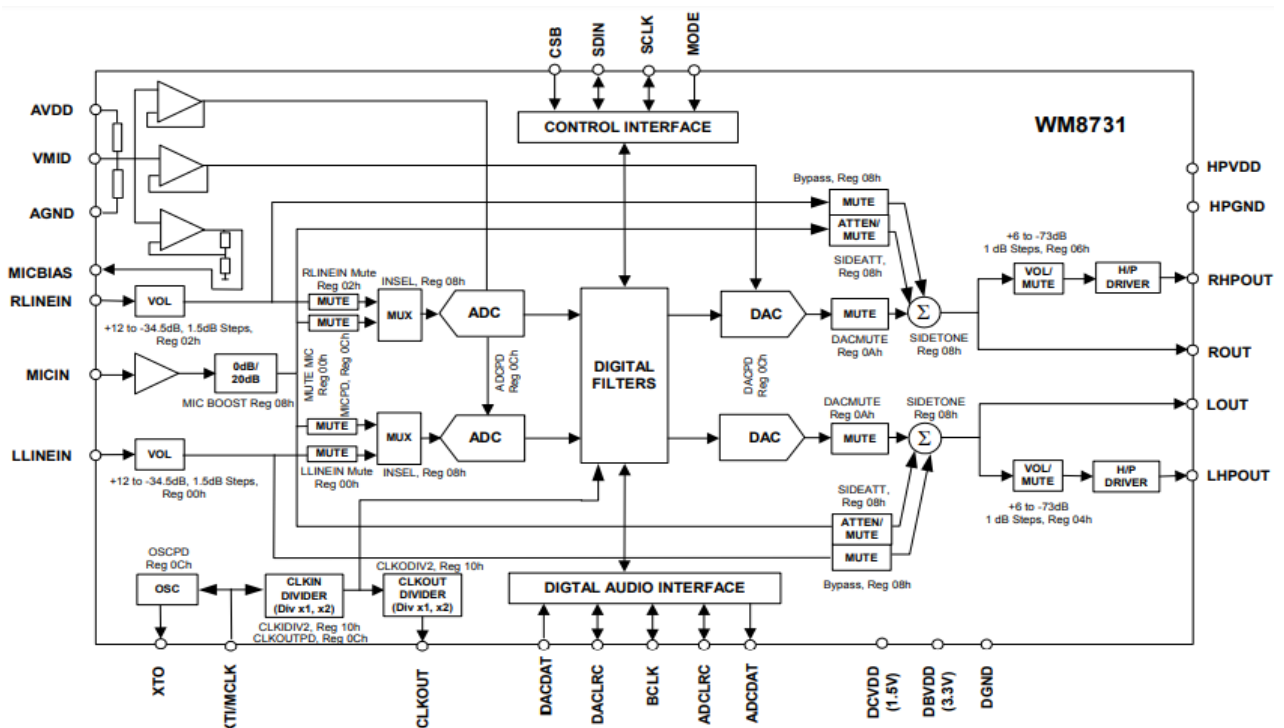


Figure 3.1 : Schéma fonctionnel du codec audio WM8731. [68]

III.2.1.1 Description des pins

Il existe 28 pins pour ce composant intégré. Le **Tableau 3.1** illustre le nom, le type et la description de chaque pin.

Tableau 3.1 : Nom, type et description de chaque pin du WM8731.[68]

Nom de pin	Type	Description
DBVDD	Supply	Digital Buffers VDD
CLKOUT	Digital Output	Buffered Clock output
BCLK	Digital Input/Output	Digital audio bit clock, pull Down, (see Note 1)
DACDAT	Digital input	DAC Digital Audio Data input
DACLRC	Digital input/Output	DAC Sample Rate Left/Right clock, Pull Down(see Note 1)
ADCDAT	Digital Output	ADC Digital Audio Data input
ADCLCR	Digital Input/Output	ADC sample Rate Left/right clock,pull Down (See Note 1)
HPVDD	Supply	Headphone VDD
LHPOUT	Analogue Output	Left Channel Headphone output
RHPOUT	Analogue Output	Right Channel Headphone output
HPGND	Ground	Headphone VDD
LOUT	Analogue Output	Left Channel line output
ROUT	Analogue Output	Right Channel line output
AVDD	Supply	Analogue VDD
AGND	Ground	Analogue GND
VMID	Analogue Output	Mid-rail reference decoupling point
MICBIAS	Analogue Output	Electret Microphone Bias
MICIN	Analogue input	Microphone input (AC coupled)
RLINEIN	Analogue input	Right channel Line input (AC coupled)
LLINEIN	Analogue input	Left Channel line input (AC coupled)
MODE	Digital input	Control interface sélection, Pull up (see Note 1)
CSB	Digital input	3-Wire MPU chip select/2-wire MPU interface Address selection, active low, Pull up (see Note 1)
SDIN	Digital Input/Output	3-Wire MPU DATA input/2-Wire MPU Data input
SCLK	Digital input	3-Wire MPU Clock input/2-Wire MPU clock input
XTI/MCLK	Digital input	Crystal input or Master clock Input (MCLK)
XTO	Digital Output	Crystal output
DCVDD	Supply	Digital Core VDD
DGND	Ground	Digital GND

III.2.1.2 Codec audio et circuit programmable

La **Figure 3.2** représente la connexion du WM8731 à un circuit programmable quelque soit la carte de développement : CPU, FPGA, µcontrôleur...

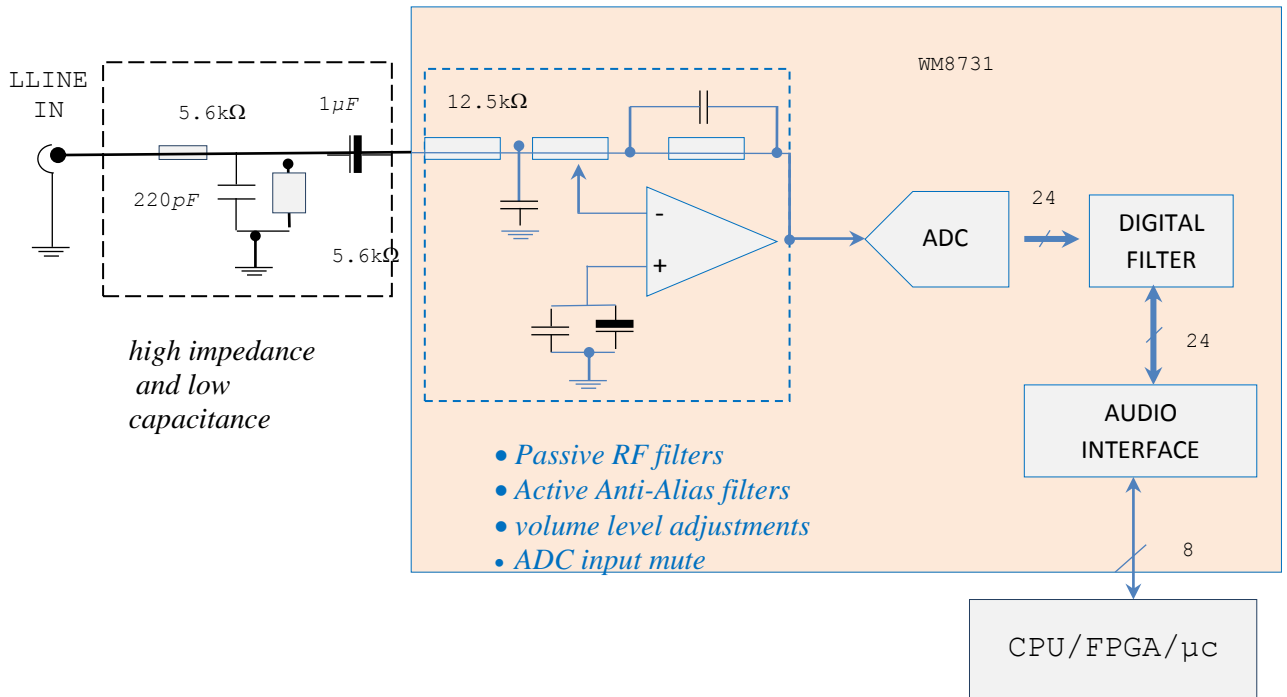


Figure 3.2: Connexion du WM8731 à un circuit programmable : CPU, FPGA, μ contrôleur...

Nous notons ce qui suit :

- Les entrées sont à haute impédance et à faible capacité, donc parfaitement adaptées à la réception de signaux Hi-Fi ou audio externes.
- L'entrée ADC pleine échelle est de 1.0 volts rms à $VDD = 3.3$ volts. La valeur RMS de l'entrée est : $rms = \frac{1}{\sqrt{2}} \times V_{peak}$
- Toute tension supérieure à la pleine échelle risque de surcharger le CAN et de provoquer une distorsion.

Sous-contrôle logiciel:

- Le gain entre les entrées ligne et l'ADC est réglable de façon logarithmique de +12 dB à -34.5 dB avec un pas de 1.5 dB.
- Les entrées ligne de l'ADC peuvent être coupées dans le domaine analogique.
- Les filtres numériques ADC contiennent un filtre passe-haut caractérisé par :

$$H(z) = \frac{1 - z^{-1}}{1 - 0.9995 z^{-1}}$$

Lorsque le filtre passe-haut est activé, le décalage CC est calculé en continu et soustrait du signal d'entrée. En réglant HPOR, la dernière valeur de décalage CC calculée est stockée lorsque le filtre passe-haut est désactivé et continuera à être soustraite du signal d'entrée. Si le

décalage CC change, la valeur stockée et soustraite ne changera que si le filtre passe-haut est activé. La **Figure 3.3** présente l'interface audio numérique avec FPGA Cyclone II.

III.2.2 Communication I2C[68]

Pour contrôler le WM8731 sur le bus 2 fils (Mode = 0), le dispositif de commande MAITRE doit initier un transfert de données en établissant une condition de démarrage, définie par une transition haut vers bas sur SDIN pendant que SCLK reste haut. Cela indique qu'une adresse et un transfert de données suivront. Tous les périphériques sur le bus 2 fils répondent à la condition de démarrage et décalent dans les huit bits suivants (adresse 7 bits + bit R/W). Le transfert est MSB d'abord. L'adresse 7 bits se compose d'une adresse de base 6 bits + un seul bit programmable pour sélectionner l'une des deux adresses disponibles pour cet appareil (0x34, 0x35). Si l'adresse correcte est reçue et le bit R/W=0, indiquant une écriture, puis le WM8731 répondra en tirant SDIN au niveau bas à l'impulsion d'horloge suivante (ACK). Le WM8731 est un périphérique en écriture seule et ne répondra qu'au bit R/W indiquant une écriture. Si l'adresse n'est pas reconnue, le périphérique retournera à l'état de repos et attendra une condition de nouveau démarrage et une adresse valide.

Une fois que le WM8731 a acquitté une adresse correcte, le contrôleur envoie huit bits de données (B15-B8). Le WM8731 accusera ensuite réception des données envoyées en tirant SDIN au niveau bas pendant une impulsion d'horloge. Le contrôleur enverra alors les huit bits de données restants (B7-B0) et le WM8731 accusera alors à nouveau en tirant SDIN bas.

Une condition d'arrêt est définie lorsqu'il y a une transition de bas en haut sur SDIN alors que SCLK est haut. Si une condition de démarrage ou d'arrêt est détectée dans le désordre à tout moment du transfert de données, l'appareil passera à l'état de repos.

Après avoir reçu une adresse complète et une séquence de données, le WM8731 retourne à l'état de repos et attend une autre condition de démarrage. Chaque écriture dans un registre nécessite la séquence complète de la condition de démarrage, de l'adresse de l'appareil et du bit R/W suivi des 16 bit d'adresse de registre et de données. La **Figure 3.4** représente le contrôleur I2C.

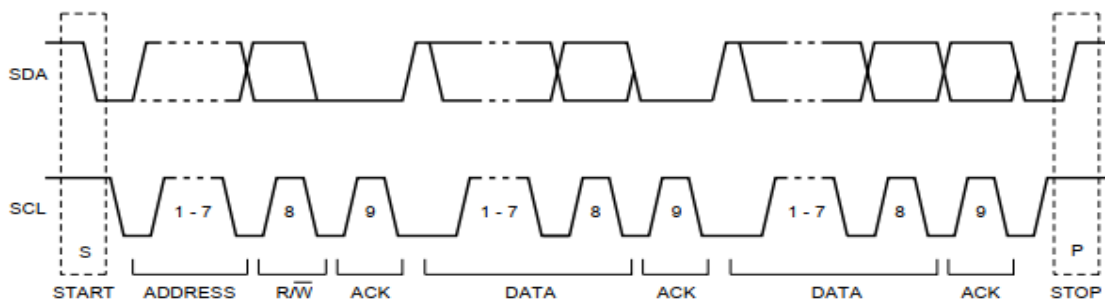


Figure 3.4 : Contrôleur I2C.

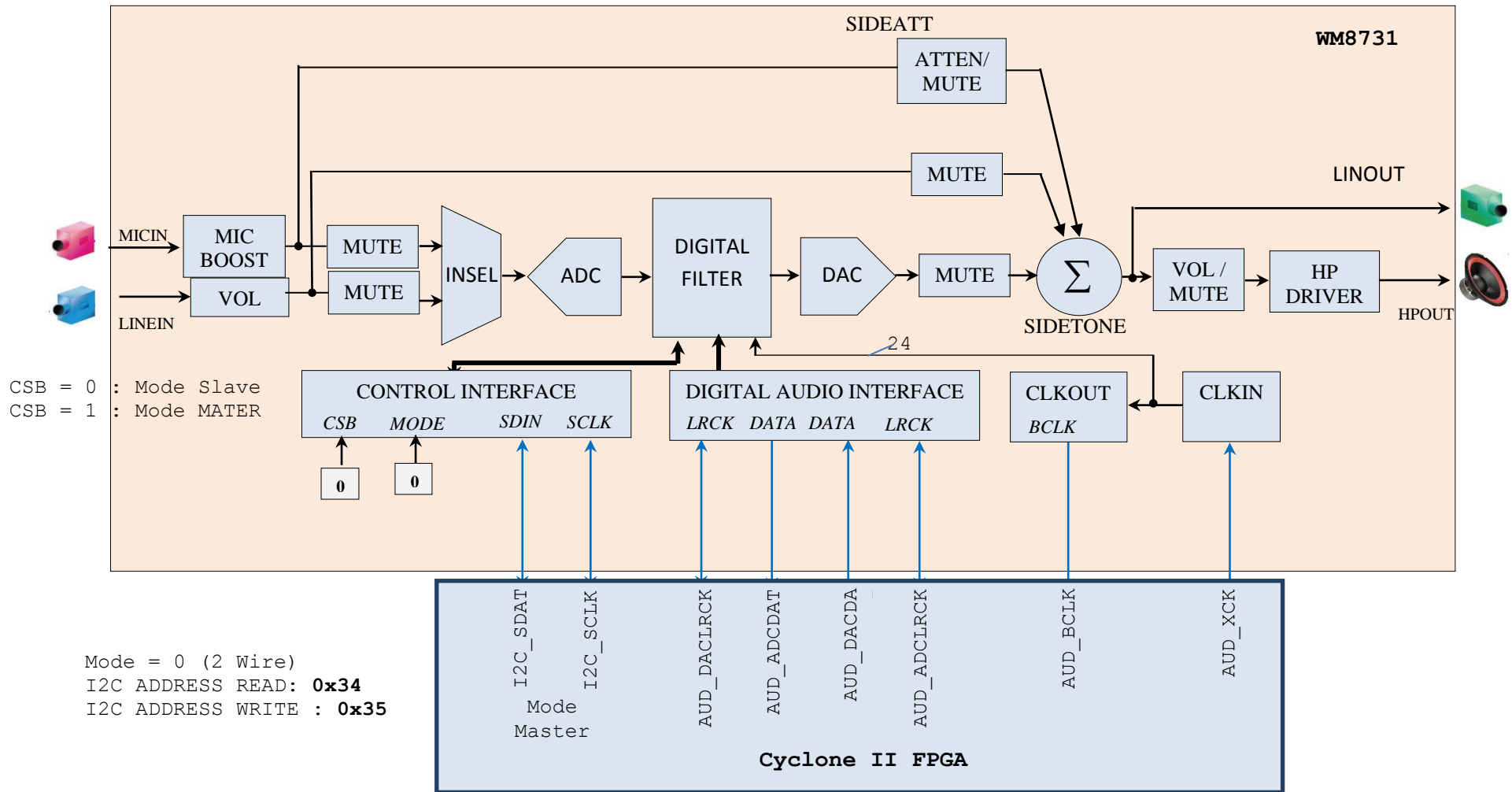


Figure 3.3: Interface audio numérique avec FPGA Cyclone II [68].

III.2.3 Adresse de chip WM8731

L'adresse du chip est constituée de 7 bits et le huitième bit c'est le bit R/W .

$8'h34 = \{7'h1A, 1'bR/W\}$ with $R/W = 0$, I2C ADDRESS Read.

$8'h35 = \{7'h1A, 1'bR/W\}$ with $R/W = 1$, I2C ADDRESS Write.

$I2C_DATA \leftarrow \{8'h34, REG_LHPOUT\};$

I2C_DAT: continent 24 Bits:

$7'bAddr_Chip + 1'bR/W + 8'bReg_Addr + 8'bDATA$

III.3 Simulations et implémentations sur FPGA

L'objectif de cette implémentation sur FPGA est de contrôler en premier lieu le son via le protocole I2C. La **Figure 3.5** illustre la configuration matérielle décrivant la méthode générale adoptée pour réaliser le circuit électrique de notre projet sur la carte de développement DE1.

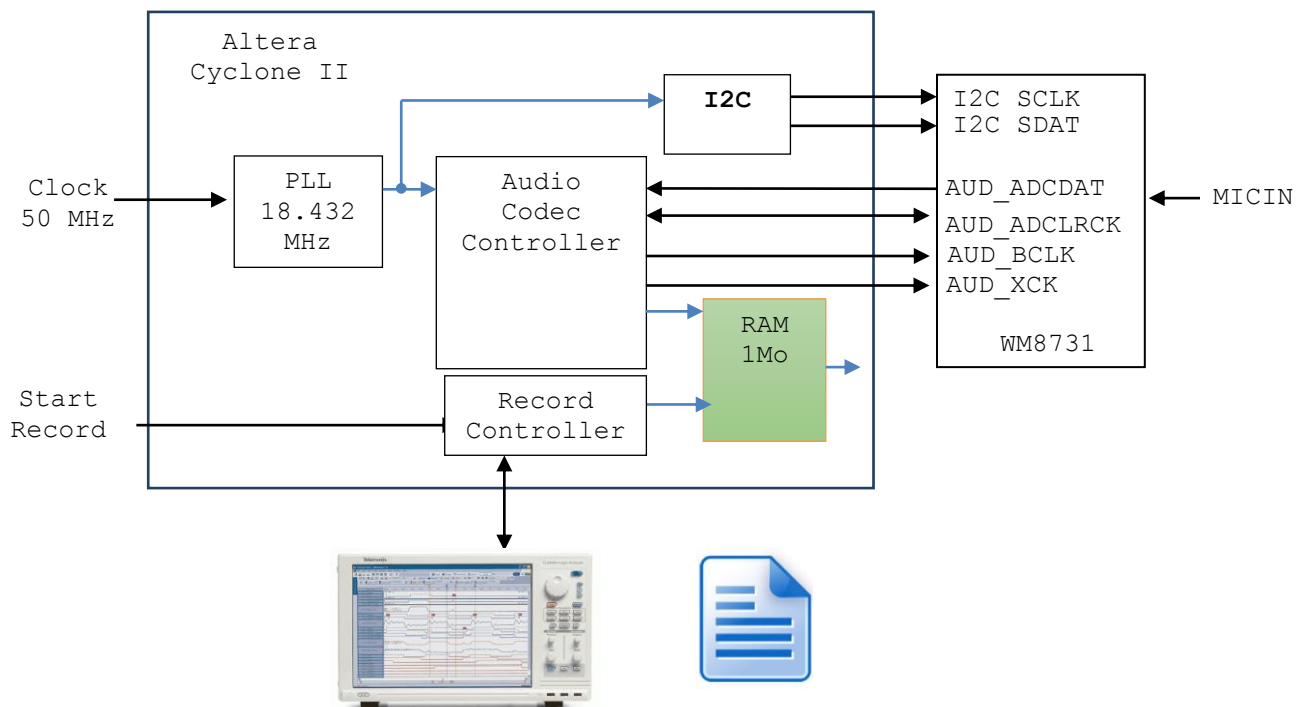


Figure 3.5 : Configuration matérielle de la méthode de contrôle de son réalisé.

III.3.1 Présentation de Quartus II [69]

La **Figure 3.6** représente l'environnement de Quartus II.

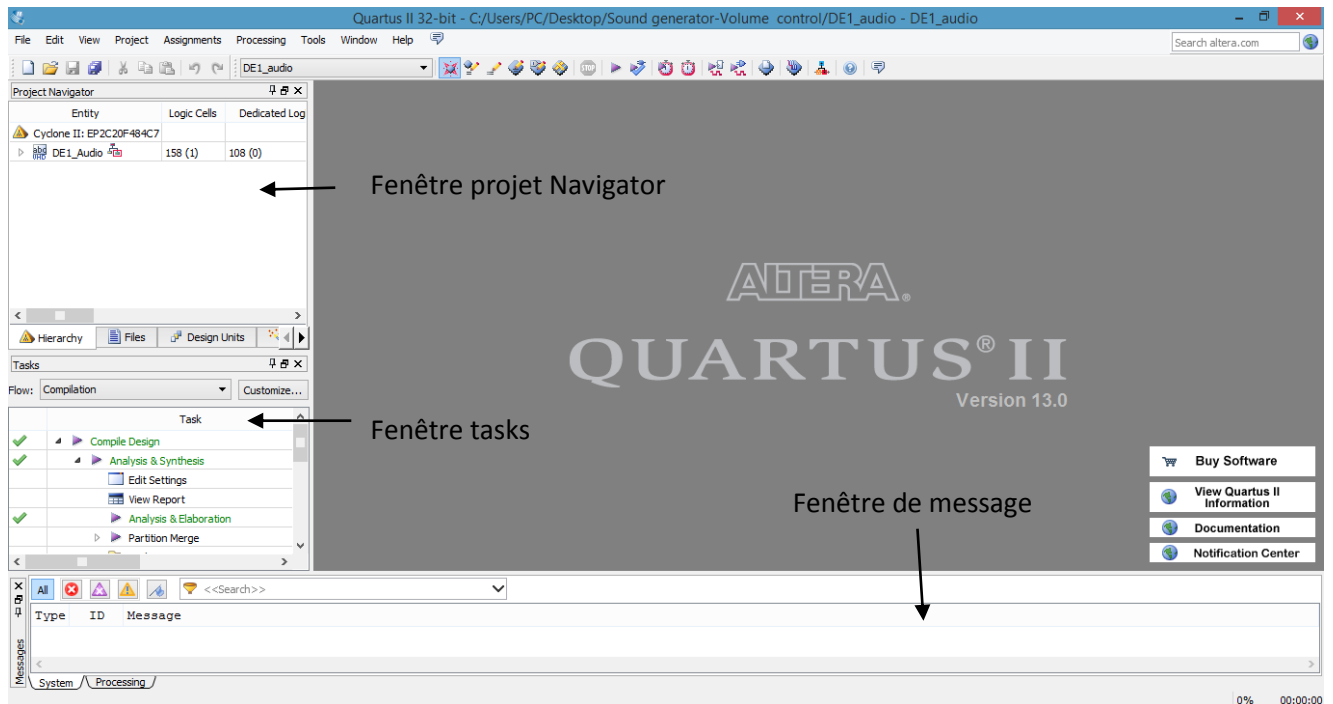


Figure 3.6 : L'environnement de développement Quartus II.


Fenêtre projet Navigator : contient toutes les informations relatives au projet dont les détails sur la hiérarchie du projet, les fichiers le constituant, aussi les schémas,

Fenêtre tasks : présente les différentes tâches avec la possibilité d'accéder à l'ensemble des processus de traitement de Quartus et les comptes rendus associés.

Fenêtre de message : permet d'être informé en continu des informations, avertissements et erreurs apparaissant lors de l'exécution des diverses tâches.

Création d'un projet sous Quartus II :

Pour créer un projet on respecte les étapes suivantes :

- 1- Entrer sur le logiciel (démarrer le Quartus II),
- 2- Créer un nouveau projet : file→ new project → next→ nom de dossier et le projet→ next → famille (cyclone II) et nom de available devices (EP2C20F484C7 pour le de1) → next→next →next→finish, comme illustré sur la **Figure 3.7**,
- 3- Ajouter une nouvelle source VHDL : file→ new →desing Files→ VHDL file (on choisit une source de type VHDL),
- 4- Sauvegarder le projet par : file→ save as
- 5- Compiler le code source par : processing → start ou par le bouton .

6- Envoyer le programme à la carte : Tools → programmer ou par le bouton  .

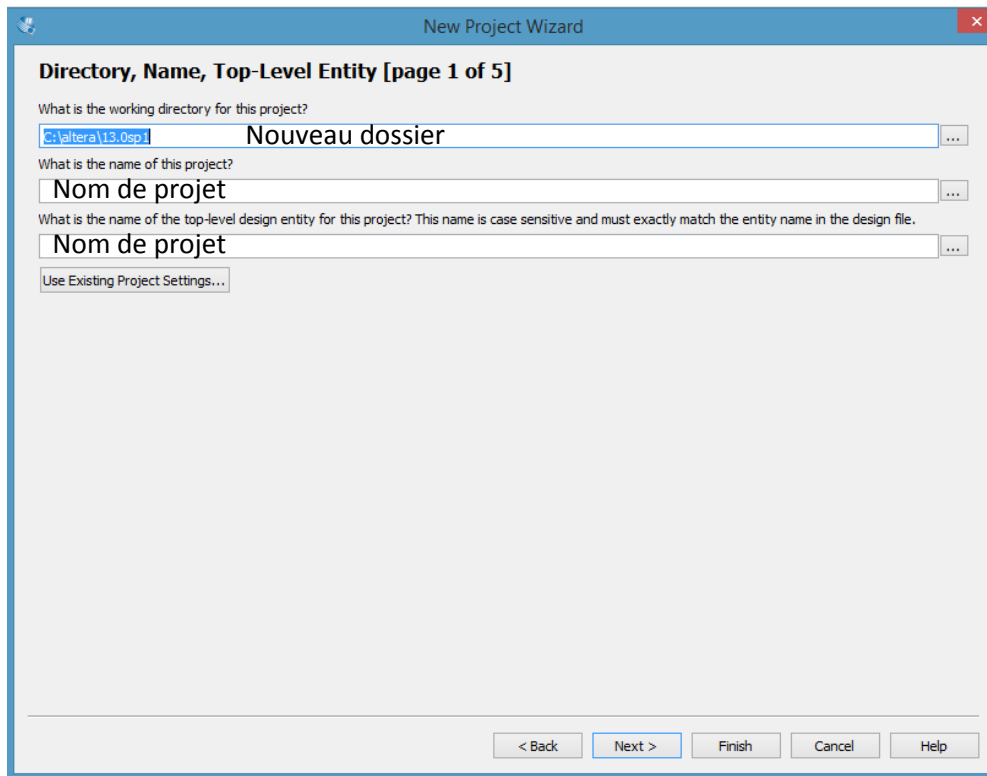


Figure 3.7 : Création d'un nouveau projet sous Quartus II [70].

Pour ouvrir le projet, on procède par : file → open Project.

III.3.2 Les étapes de simulation

- 1) Nous voulons implémenter un contrôleur I2C dans une FPGA de type Cyclone II selon les programmes VHDL suivants:

DE1_Audio.vhd (programme principale)

- audio_pll.vhd
- DATA_Rom.vhd
- codec.vhd
- I2C.v

Les codes source détaillés sont fournis dans l'Annexe A.

- 2) Après l'implémentation de notre programme sur FPGA, nous avons consommé les sources suivantes :

a- Audio_PLL

C'est un contrôleur de fréquence qui synchronise l'horloge d'entrée afin de générer l'horloge de sortie. Son rôle est de diviser la fréquence.

Illustré sur la Figure 3.8 le circuit logique de l'Audio_PLL : inclk présente l'entrée et c0 présente la sortie.



Figure 3.8: Le circuit logique de l'audio_PLL [70].

b- DATA_ROM

La Rom est une mémoire morte pour lecture seulement. Comme illustré sur la Figure 3.9, elle a trois ports : clock et addr (provenant du nœud de sortie de WM8731) pour les entrées et data_from_rom pour la sortie. La configuration matérielle de la DATA_ROM se fait en implémentant DATA_ROM.vhd sur FPGA.

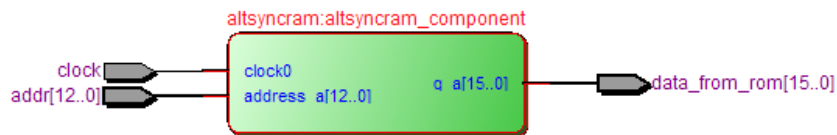


Figure 3.9: data rom [70].

On utilise DATA_ROM et Audio_pll autant que contrôleur du Codec_WM8731.[69]

c- Codec_WM8731

C'est une interface de contrôle, son rôle est de contrôler le son comme illustré à la Figure 3.10 à travers 11 portes d'entrées/sorties. Le programme VHDL Codec_WM8731.vhd est implémenté sur FPGA.

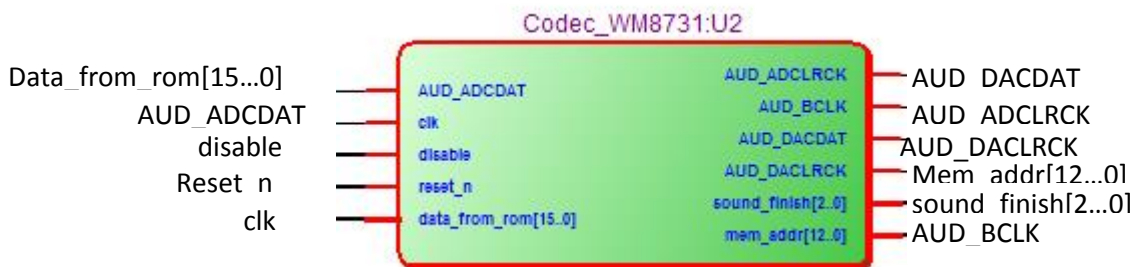


Figure 3.10 : Codec_WM8731 [70].

d- Câblage I2C

C'est un contrôleur de communication, celui qui contrôle les circuits électriques et l'application. Son circuit électronique est illustré sur la Figure 3.11.

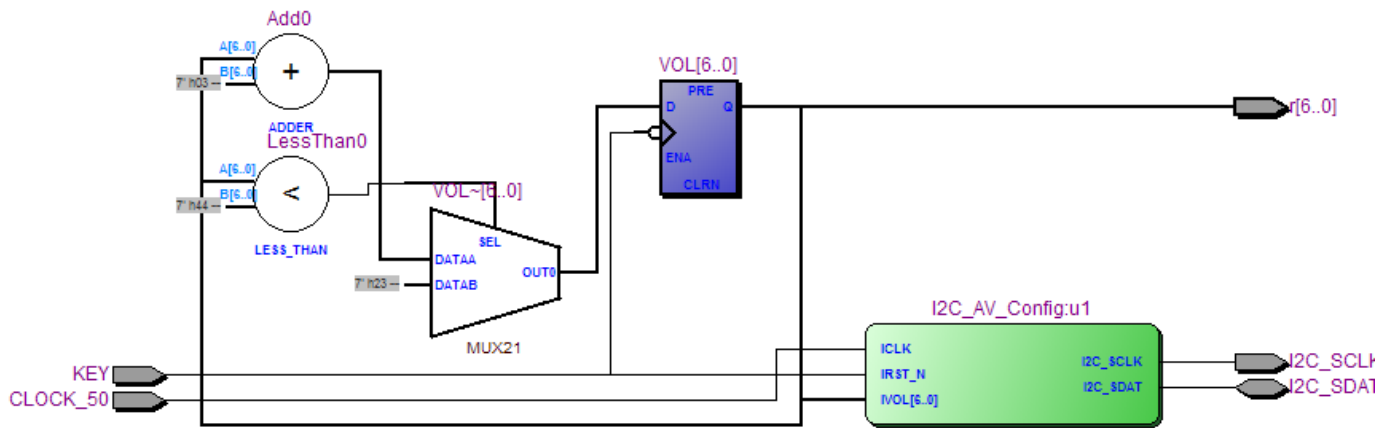


Figure 3.11: Câblage I2C [70].

Contrôle de volume:

Les différentes configurations relatives au contrôle de volume sont affichées sur les Tableaux 3.2-3.3.

Tableau 3.2: Left and Right Channel Line Input (LLINEIN & RLINEIN).

Dec	HEX	Bin	dB	step
[23..31]	[17..1F]	[101110..11111]	0dB à +12dB	1.5 dB
23	17	10111	0dB	
[0..23]	[0..17]	[00000..101110]	-34.5dB à 0dB	

Tableau 3.3: Left and Right Channel Headphone Output (LHPOUT & RHPOUT).

Dec	HEX	Bin	dB	step
[121..127]	[79..7F]	[1111001..1111111]	0dB à +6dB	1dB
121	79	1111001	0dB	
[48..121]	[30..79]	[0101111..0110000]	-73db à 0dB	
[0.. 47]	[0..2F]	[0000000..0101111]	MUTE	

Adresse du registre LLINEIN (h02) (Figure 3.12)

0	0	0	0	0	1	0	0	0	1	1	1	1	0	0	1
Address Register							LRHPBOTH	LZCEN	LHPVOL [6:0]						

Figure 3.12 : Adresse du registre LLINEIN.

```
REG_LHPOUT<= {7'h02, 1'b0, 1'b0, LHPVOL};
```


Après avoir jointre tous les programmes avec le programme principal DE1_Audio.vhd, nous construisons le circuit électronique final comme reporté sur la **Figure 3.13**.

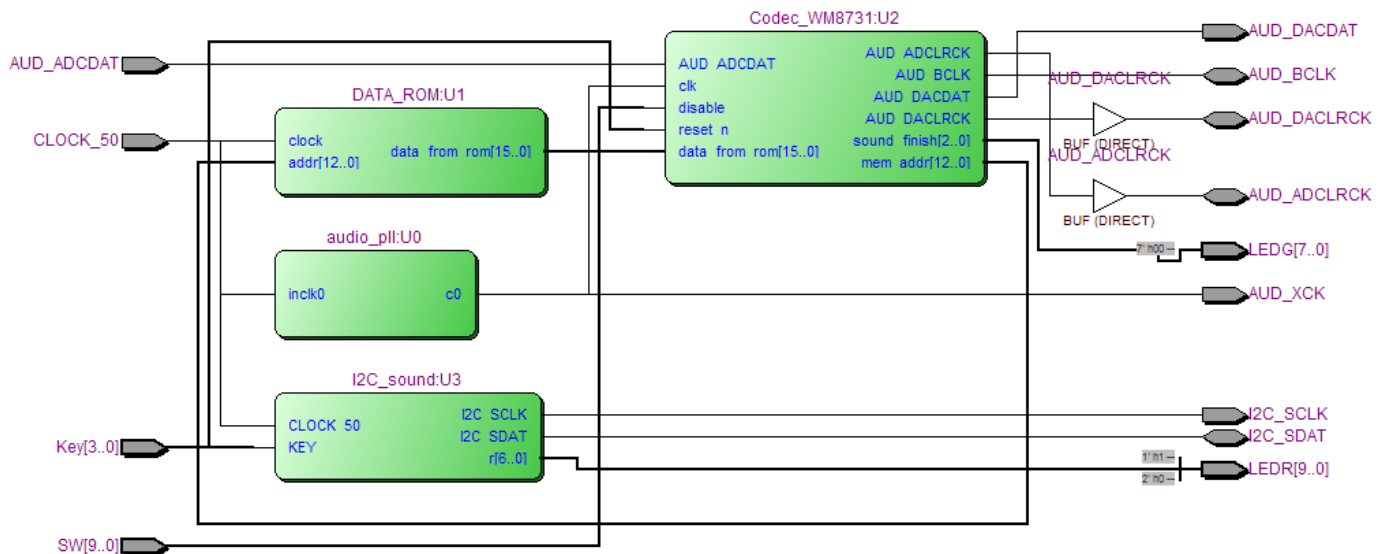


Figure 3.13: Schéma global de la FPGA et le codec audio WM8731 [70].

III.4 Manipulations sur le projet réalisé

Nous avons assemblé le PC avec la carte de développement DE1 et les baffles comme illustré sur la **Figure 3.14** suivante :

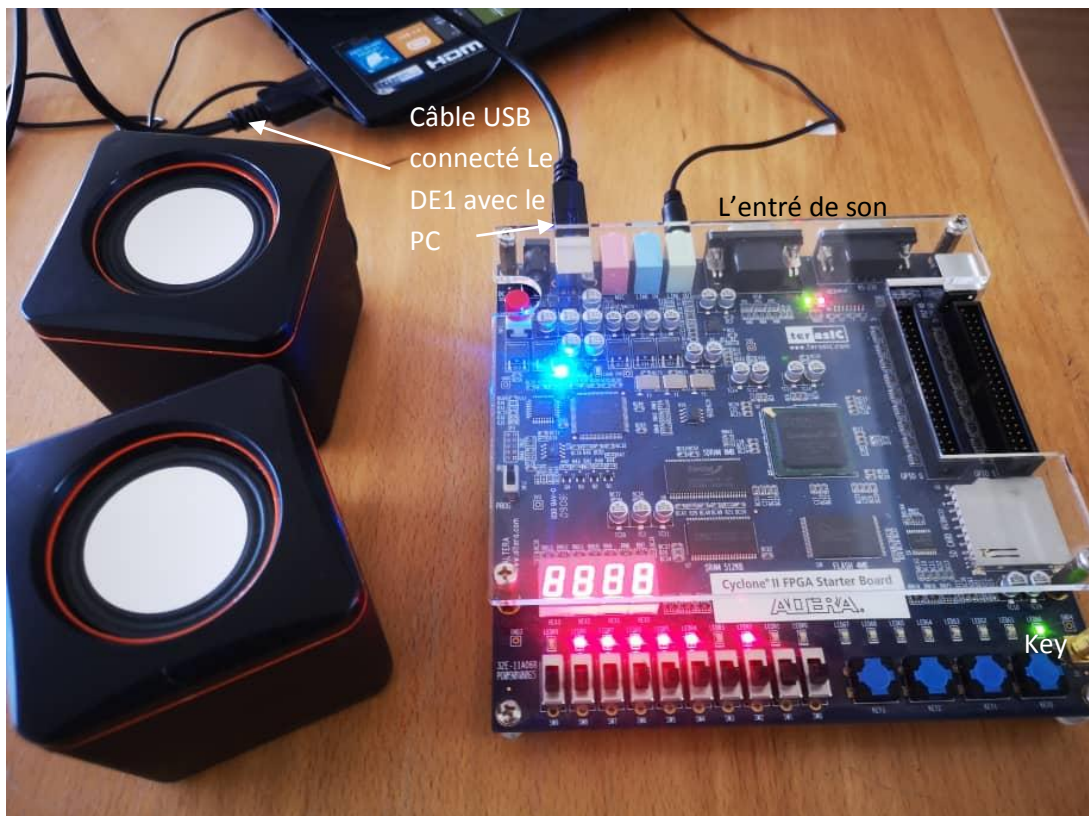



Figure 3.14 : Photo représentant les manipulations.

Après la compilation, nous envoyons le programme sur la carte en passant par : programmer , puis Start comme le montre la **Figure 3.15**. Et à partir de là, nous pouvons contrôler le volume de son.

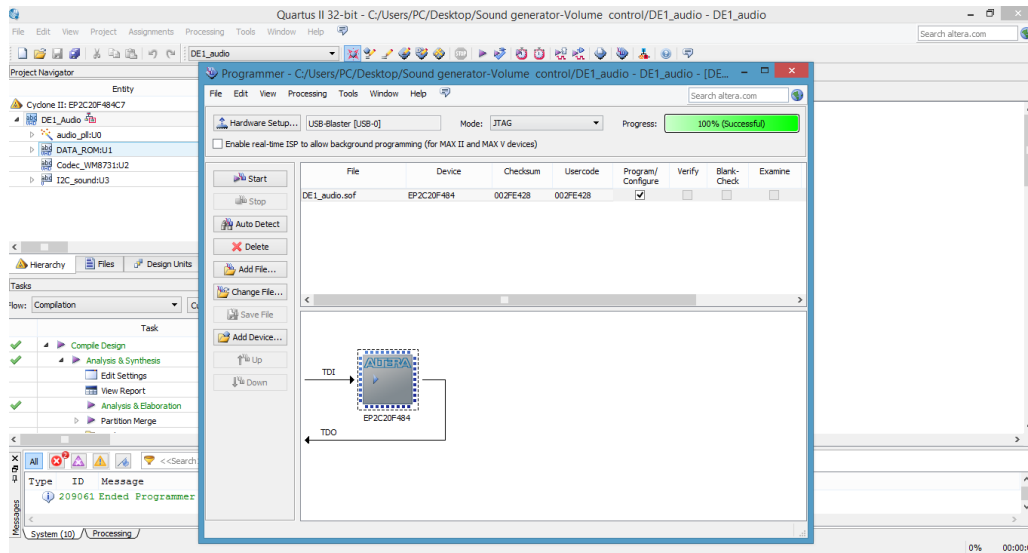


Figure 3.15: Envois du code source à la carte DE1 après montage des différents composants et connectiques [70].

III.5 Conclusion

Dans ce chapitre nous avons réussi à implémenter matériellement le protocole de communication I2C sur une carte de développement DE1 en utilisant le langage VHDL. Comme application pratique, nous avons pu contrôler le codec audio WM8731 via la carte FPGA ALTERA Cyclone II. Ces deux modules sont intégrés sur la carte DE1. Ceci nécessite la rédaction d'un paquet de codes sources relatifs aux différentes tâches, entre autres, un programme de mémoire contenant le fichier son, des programmes configurant la communication série par le protocole I2C et le codec audio WM8731 ainsi que des programmes de configuration des différentes entrées/sorties y compris les baffles pour exploiter le signal audio contrôlé.

Le lecteur est sollicité à lire l'Annexe A pour voir le paquet entier des programmes en VHDL qui nous a permis de réaliser notre projet de contrôle du codec audio via le protocole I2C en utilisant une carte FPGA.

Conclusions et recommandations pour les travaux futurs

I. Conclusions

Pour réaliser ce projet, nous avons essayé d'utiliser nos connaissances acquises durant la durée de nos études dans la faculté des sciences et de la technologie, quelques recherches sur Internet et bien évidemment, l'aide de nos professeurs encadrant.

Dans le contexte du présent projet, l'objectif principal était de réaliser un protocole de communication I2C pour contrôler le codec audio WM8731. Pour y parvenir, nous avons utilisé la carte de développement DE1 qui contient les modules : Le **WM8731** et le circuit programmable **FPGA ALTERA Cyclone II**.

Nous avons réalisé une interface du contrôle du volume d'un son dans le codec audio WM8731 permettant, en premier lieu, d'effectuer les tâches suivantes : Augmenter/diminuer le volume, muter/écouter et démarrer/arrêter.

Parmi les divers protocoles disponibles décrits dans l'état de l'art du chapitre 1, nous avons choisi d'utiliser le protocole I2C pour réaliser une communication série entre la FPGA, le module de contrôle, et le WM8731, le module contrôlé. Le développement des programmes VHDL nécessaires à notre implémentation matérielle était réalisé sous l'environnement **Quartus ALTERA V 13.0**.

Au cours de l'achèvement du projet, nous avons rencontré des obstacles et des problèmes qui ont retardé son achèvement. La première entrave provient du fait que, pendant cette année très exceptionnelle, nous avons commencé nos travaux de projet de fin d'études très tard. La deuxième entrave était la non-possession d'un PC de qualité qui facilitera nos travaux sachant que le logiciel Quartus ALTERA V 13.0 est gourmand en mémoire. Ce problème générique a beaucoup bloqué les étudiants du M2 surtout que les prix des PC ont augmentés d'une façon incroyable. Par cette occasion, nous demandons à la tutelle de penser à aider les étudiants à avoir leurs propres PCs. La troisième entrave concerne les coupures exagérées de l'internet qui nous ont beaucoup retardées. Cette situation a persisté en particulier lors de l'examen du baccalauréat.

II. Recommandations

Notre projet peut être développé dans plusieurs applications pédagogiques et industrielles. Nous recommandons les personnes intéressées de développer le présent travail dans les axes suivants:

- Développement de travaux pratiques relatifs aux technologies et protocoles multimédia pour les étudiants en master télécommunications et aussi systèmes embarqués
- Réalisation de réseaux à base de FPGA incluant plusieurs utilisateurs
- Considération d'autres types de protocoles pour établir des communications entre différentes entités,
- Etude d'autres protocoles de communications audio et vidéo pour applications multimédia avancées à savoir la télévision numérique, téléphone IP, etc.

Bibliographie :

- [1] François Goffinet, Protocoles et modèles de communication, 10 août 2020. Disponible sur : <https://cisco.goffinet.org/ccna/fondamentaux/protocoles-modeles-communication/>
- [2] Low-voltage differential signalling, Disponible sur : https://en.wikipedia.org/wiki/Low-voltage_differential_signaling. 2021
- [3] <https://www.micro-epsilon.fr/service/glossar/LVDS.html>
- [4] Conal Watterson, LVDS and M-LVDS Circuit Implementation Guide 2013, Disponible sur : <https://www.analog.com/media/en/technical-documentation/application-notes/AN-1177.pdf>
- [5] Marc Defossez. "D-PHY Solutions", Disponible sur : https://www.xilinx.com/support/documentation/application_notes/xapp894-d-phy-solutions.pdf
- [6] Multipoint-Low Voltage Differential Signaling (M-LVDS) Evaluation Module, User's Guide, Texas Instruments, June 2007. Disponible sur : https://www.ti.com/lit/ug/sllu094/sllu094.pdf?ts=1624216962306&ref_url=https%253A%252F%252Fwww.google.com%252F
- [7] Use Flat Panel Display (FPD) Link Converters for Video Streaming, BY OCTOPART, FRI 18 January 2019, Disponible sur: <https://octopart.com/blog/archives/2019/01/flat-panel-display-fpd-link-converters-video-streaming>
- [8] FDP, <https://en.wikipedia.org/wiki/FPD-Link>
- [9] Cours Systèmes Embarqués: Le Bus CAN, Révisé le : 18-11-2017, disponible sur : <https://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-CAN.htm>
- [10] Z. MAMMERI – UPS, Introduction au réseau CAN, disponible sur : https://www.irit.fr/~Zoubir.Mammeri/Cours/Introduction_CAN.pdf
- [11] Eric DELAUNAY, le bus CAN, disponible sur: <https://www.lelectronique.com/dossier/le-bus-can-7.html>
- [12] HADJI Djamel et TOUCHERIFT Hacene , Conception et réalisation d'un système de diagnostic et géolocalisation d'un véhicule, master en réseaux mobilités et système Embarqués, Université Mouloud Mammeri de Tizi-Ouzou, Faculté Génie Electrique et d'informatique, Département d'informatique, 2016/2017. Disponible sur : <https://dl.ummto.dz/bitstream/handle/ummto/12466/HadjiDjamel.pdf?sequence=1&isAllowed=y>
- [13] LEÇON 8-CAPTEURS I2C, PHAROIoT Plateforme de programmation en direct. Disponible sur : <https://www.pharoiot.org/lesson-8-i2c-sensors/>

- [14] LIN (LOCAL INTERCONNECT NETWORK) SOLUTIONS by Microcontroller Division Applications, Disponible sur:
https://www.st.com/resource/en/application_note/cd00004273-lin-local-interconnect-network-solutions-stmicroelectronics.pdf
- [15] Introduction to the Local Interconnect Network (LIN) Bus, Updated Oct 14, 2020, Disponible sur :
<https://www.ni.com/en-lb/innovations/white-papers/09/introduction-to-the-local-interconnect-network--lin--bus.html>
- [16] Clemson Vehicular Electronics Laboratory: LIN Bus, LIN (réseau d'interconnexion local). Disponible sur:
<https://cecas.clemson.edu/cvel/auto/Buses/LIN.html>
- [17] Quelle est la différence entre le DVI-I et le DVI-D ?,demolinux. Disponible sur:
<https://demolinux.org/quelle-est-la-difference-entre-le-dvi-i-et-le-dvi-d/>
- [18] Digital Visual Interface (DVI). Disponible sur :
<https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W120623D.pdf>
- [19] philippe, Génération d'une trame DVI avec un FPGA, octobre 12, 2018, Disponible sur :
<https://systemes-embarques.fr/wp/archives/generation-dune-trame-dvi-avec-un-fpga/>
- [20] ANIKET MANI TRIPATHI, HIGH DEFINITION MULTIMEDIA INTERFACE (HDMI), Cochin University of Science & Technology Kochi-682022, OCTOBER 2010, Disponible sur :
<http://dspace.cusat.ac.in/jspui/bitstream/123456789/2212/1/HDMI.pdf>
- [21] HDMI Transmitter (HDMI),Freescale Semiconductor, Inc, i.MX 6Dual/6Quad Applications Processor Reference Manual, Rev. 2, 06/2014, Disponible sur :
<https://people.freebsd.org/~gonzo/arm/iMX6-HDMI.pdf>
- [22] BY KAELIN LEAVITT, HDMI Cable Specification Considerations,JANUARY 22, 2017, Disponible sur :
<https://summitech.com/2017/01/22/hdmi-cable-specification-considerations-spring-2017/>
- [23] ANSCHLUSS-WIRRWARR: HDMI VS DISPLAYPORT VS DVI VS USB-C VS VGA, Disponible sur :
<https://gaming-kaufberater.de/hdmi-vs-dp-vs-dvi-vs-usb-c-vs-vga/>
- [24] Prise HDMI, Disponible sur :
<https://askfrance.me/q/pourquoi-bouger-les-pistes-a-proximite-sur-un-pcb-59836608610>
- [25] VESA DisplayPort Standard Version 1, Revision 2 January 5, 2010. Disponible sur :
<https://glenwing.github.io/docs/DP-1.2.pdf>

[26] Craig Wiley, DisplayPort Technical Overview IEEE International Conference on Consumer Electronics (ICCE) Advances & Challenges in HD Interconnects January 10, 2011 Las Vegas. Disponible sur :

<file:///E:/reff%C3%A9rence/ICCE-Presentation-on-VESA-DisplayPort.pdf>

[27] Le DisplayPort, à quoi ça sert ?, Connectique et Infrastructure Réseau, jlgdiscount ,5 décembre. Disponible sur :

<https://www.jlgdiscount.fr/blog/2016/12/le-displayport-a-quoi-ca-sert/>

[28] Introduction to HART, Emerson Process, ©2002. Disponible sur :

<https://www.emerson.com/documents/automation/training-introduction-to-hart-en-41098.pdf>

[29] What is HART Protocol?, Disponible sur :

<https://instrumentationtools.com/what-is-hart-protocol/>

[30] Tutoriel de communication HART, partie 3. Disponible sur :

<https://instrumentationtools.com/hart-communication-tutorial-part-3/>

[31] Eric Peña and Mary Grace Legasp, UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter, analog dialogue, 4 December 2020. Disponible sur :

<https://www.analog.com/media/en/analog-dialogue/volume-54/number-4/uart-a-hardware-communication-protocol.pdf>

[32] Patrice KADIONIK, LE BUS INDUSTRIEL PCI, ENSEIRB école national supérieure électronique informatique & Radiocommunication. Disponible sur :

https://kadionik.vvv.enseirb-matmeca.fr/formation/bus_industriels/pci_enseirb.pdf

[33] Yves Joskin, R&D Manager, Technology Note PCI Bus Variation, Copyright 2006, Disponible sur :

https://documentation.euresys.com/Products/MultiCam/MultiCam_6_16/Content/MultiCam_6_7_HTML_Documentation/PCI_Bus_Variation.pdf

[34] TOM SHANANLEY and DON ANDERSON, PCI system Architecture fourth edition, MINSHARE.INC, ISBN:0-201-30974-2, may 1999. Disponible sur :

[file:///E:/reff%C3%A9rence/PCI%20System%20Architecture%20\(4th%20Edition\)\(%20data\).pdf](file:///E:/reff%C3%A9rence/PCI%20System%20Architecture%20(4th%20Edition)(%20data).pdf)

[35] Par Stephen J. Bigelow, rédacteur technologique principal. Disponible sur :

<https://searchdatacenter.techtarget.com/definition/PCI-Express>

[36] Jean-François Pillou, Bus PCI Express (PCI-E), mercredi 19 octobre 2016. Disponible sur :

<https://www.commentcamarche.net/contents/757-bus-pci-express-pci-e>

[37] <https://www.eeterms.com/terms/p.html>

[38] Robert Murphy, USB 101: An Introduction to Universal Serial Bus 2.0 A, Cypress, Document No. 001-57294 Rev. *H, Disponible sur :

<https://www.cypress.com/file/134171/download>

[39] Bus Universel en série (USB-universal sériel bus) ,27/05/2014, Disponible sur :

<https://slideplayer.fr/slide/3018268/>

[40] Riadh Hajji, types d'interface des disques durs, 6mars2018. Disponible sur :

<https://apcpedagogie.com/types-dinterface-des-disques-durs/>

[41] Understanding and Performing MIPI® D-PHY Physical Layer, CSI and DSI Protocol Layer Testing. Disponible sur :

https://download.tek.com/document/61W_25772_0_HR.pdf

[42] Le bus CSI et Le bus DSI du Raspberry Pi 3 B+, Disponible sur :

<https://www.editions-eni.fr/open/mediabook.aspx?idR=d8d69e9bd6ae26780c54d31950c21a79>

[43] Various display interfaces DBI, DPI, LTDC, DSI, FSMC, Disponible sur :

<https://www.programmingsought.com/article/82725633669/>

[44] MIPI Display Serial Interface (DSI). Disponible sur :

<https://focuslcds.com/mipi-display-serial-interface-dsi/>

[45] KeyStone Architecture Literature Serial Peripheral Interface (SPI),Texas instrument, March 2012. Disponible sur :

<https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf>

[46] Pavel Bohá ik, Périphérique série MPC5121e Interface (SPI), Freescale Semiconductor, Rev. 0, 08/2009. Disponible sur :

<https://www.nxp.com/docs/en/application-note/AN3904.pdf>

[47] Nirav Desai (Ingénieur de conception FPGA (R&D), SATA Protocol,13 juin 2014.

Disponible sur :

<https://www.slideshare.net/niravdesai7121/sata-protocol>

[48] Serial attached SCSI. Disponible sur :

<https://www.pcmag.com/encyclopedia/term/serial-attached-scsi>

[49] protocole SAS, 26 octobre 2014. Disponible sur :

<https://fdocuments.in/document/sas-protocol.html>

[50] Mike Jackson, SAS Storage Architecture, MINDSHARE PRESS, revision1.1, Disponible sur :

<https://www.mindshare.com/files/ebooks/SAS%20Storage%20Architecture.pdf>

[51] Système IO-Link, Description fonctionnelle, 01/2013, A5E31637729-AA. Disponible sur :

https://media.automation24.com/manual/fr/io_link_system_function_manual_fr-FR_fr-FR.pdf

[52] Jeff DeAngelis Managing Director Industrial & Healthcare Business, Unit IO-Link Handbook, Maxim integrated. Disponible sur :

<https://pdfserv.maximintegrated.com/en/sg/IO-Link-Handbook.pdf>

[53] Bus Ethernet, 13/06/15. Disponible sur :

http://www.interfacebus.com/Design_Connector_Ethernet.html

[54] Ethernet Bus Network Topology, Disponible sur:

<https://aeondtech.blogspot.com/2016/11/ethernet-bus-network-topology.html>

[55] MEHDI NAIT ALAMARA , Mémoire de fin d'études d'ingénieur en Electronique, Mesure de la température du sol à base du bus de communication IIC , UNIVERSITE MOULOU MAMMERI, TIZI-OUZOU-2010

[56] Disponible sur : <https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/>

[57] Disponible sur : <https://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-bus-I2C.htm>

[58] Disponible sur : <https://docplayer.fr/47620114-Protocole-i2c-i2c-caracteristiques-principales-i2c-cablage-i2c-glossaire.html>

[59] Disponible sur : www.suaentcompani

[60] Disponible sur : <https://peitecquery.wixsite.com/arduino-passion/la-liaison-i2c>

[61] Disponible sur : <https://www.aurel32.net/elec/i2c.php>

[62] Disponible sur : <https://www.hispavila.com/introduccion-al-i2c/>

[63] Aberkane Mounir, Mémoire de Fin d'Etude, Mise en œuvre du protocole I2C Dans un environnement à microcontrôleur Microchip (PIC16F877), Université Mohamed Khider Biskra ,2013

[64] Disponible sur : <https://hackaday.com/2015/06/25/embed-with-elliott-i2c-bus-scanning/>

[65] Disponible sur : <https://www.emi.ac.ma/oumnad/ARDUINO/arduino.html>

[66] Disponible sur : <http://electroniqueamateur.blogspot.com/2020/01/analyse-dune-communication-i2c.html?m=1>

[67] Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates, WOLFSON MICROELECTRONICS plc, Production Data, February 2005, Rev 4.0, Disponible sur :

<http://cdn.sparkfun.com/datasheets/Dev/Arduino/Shields/WolfsonWM8731.pdf>

[68] Hadj Said Djamel, Implémentation d'une application de tracking sur FPGA, master II en Electronique option Réseaux et Télécommunication, UNIVERSITÉ MOULOU MAMMERI TIZI-OUZOU Faculté de Génie Électrique et D'informatique Département d'électronique, 2012-2013

[69] Logiciel Quartus ALTERA V 13.0

Annex

Annex A

Programmes :

DE1_Audio.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity DE1_Audio is port (
  CLOCK_50   : in  std_logic;           -- 50 MHz clock
  LEDR       : out std_logic_vector(9 downto 0); -- Red LEDs
  LEDG       : out std_logic_vector(7 downto 0); -- Red LEDs
  SW         : in  std_logic_vector(9 downto 0);
  Key        : in  std_logic_vector(3 downto 0);
  -- Audio CODE WM8731
  AUD_ADCLRCK : inout std_logic;       -- ADC LR Clock
  AUD_ADCDAT  : in  std_logic;         -- ADC Data
  AUD_DACLK   : inout std_logic;       -- DAC LR Clock
  AUD_DACDAT  : out std_logic;         -- DAC Data
  AUD_BCLK    : inout std_logic;       -- Bit-Stream Clock
  AUD_XCK     : out std_logic;         -- Chip Clock
  -- I2C for CODE WM8731
  I2C_SDAT    : inout std_logic;       -- I2C Data
  I2C_SCLK    : out std_logic );       -- I2C Clock

End DE1_audio;

Architecture rtl of DE1_Audio is
-----
-- PLL : 50 MHz -----> 18.432 MHz
-----
Component audio_pll IS      PORT (
  inclk0 : IN STD_LOGIC := '0';
  c0      : OUT STD_LOGIC );
End component;
-----
-- Audio CODE WM8731
-----
Component Codec_WM8731 is port (
  clk           : in std_logic;
  reset_n       : in std_logic;
  disable       : in std_logic;
  data_from_rom : in std_logic_vector(15 downto 0);
  sound_finish  : out std_logic_vector(2 downto 0);
  mem_addr      : out unsigned(12 downto 0);

  AUD_ADCLRCK : out std_logic;
  AUD_ADCDAT  : in  std_logic;
  AUD_DACLK   : out std_logic;
  AUD_DACDAT  : out std_logic;
  AUD_BCLK    : inout std_logic );
End component;
-----
-- I2C : configuration of WM8731CODE
```

Annex

```

-----
Component I2C_sound IS      PORT (
  CLOCK_50 : IN STD_LOGIC ;          --      50 MHz
  KEY      : IN STD_LOGIC ;          --
  Pushbutton[3:0]
    I2C_SDAT : inout  std_logic;      --//   I2C Data
    I2C_SCLK : out   std_logic;      --//   I2C Clock
    r       : out   STD_LOGIC_VECTOR (7 DOWNT0 0) );

End component;
-----
-- ROM : sound
-----
Component DATA_ROM IS      PORT (
  addr      : IN unsigned (12 DOWNT0 0);
  clock     : IN STD_LOGIC ;
  data_from_rom : OUT STD_LOGIC_VECTOR (15 DOWNT0 0) );

End component;

signal audio_clock_18 : std_logic;
signal temp : std_logic_vector(3 downto 0);
signal mem_addr_rom01 : unsigned(12 downto 0);
signal data_ROM1 : STD_LOGIC_VECTOR (15 DOWNT0 0);

Begin

U0 : audio_pll port map(
  inclk0 => CLOCK_50,
  c0     => audio_clock_18);

U1 : DATA_ROM port map (
  addr      => mem_addr_rom01,
  clock     => CLOCK_50,
  data_from_rom => data_ROM1 );

U2 : Codec_WM8731 port map (
  clk           => audio_clock_18, -- Audio CODEC Chip Clock AUD_XCK (18.43
MHz)
  reset_n      => Key(0),
  disable      => SW(0),
  data_from_rom => data_ROM1,
  sound_finish(0) => LEDG(0),
  mem_addr     => mem_addr_rom01,
  --Audio interface signals
  AUD_ADCLRCK => AUD_ADCLRCK,  -- out Audio CODEC ADC LR Clock
  AUD_ADCDAT  => AUD_ADCDAT,   -- in  Audio CODEC ADC Data
  AUD_DACLCK  => AUD_DACLCK,   -- out Audio CODEC DAC LR Clock
  AUD_DACDAT  => AUD_DACDAT,   -- out Audio CODEC DAC Data
  AUD_BCLK    => AUD_BCLK );  -- inout Audio CODEC Bit-Stream Clock

U3: I2C_sound port map (
  CLOCK_50 => CLOCK_50,          --      50 MHz

```

Annex

```
KEY          => Key(0),          -- Pushbutton[3:0]
I2C_SDAT => I2C_SDAT,          -- I2C Data
I2C_SCLK => I2C_SCLK , -- I2C Clock
r          => LEDR(9 downto 2));

AUD_XCK <= audio_clock_18;
temp  <= SW(9)&SW(8)&SW(7)&SW(6);

LEDG( 7 downto 1) <= "0000000";
LEDR (1 downto 0) <= "00";

End;
```

Audio_pll.vhd

```
-- megafunction wizard: %ALTPLL%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altpll

-- =====
-- File Name: audio_pll.vhd
-- Megafunction Name(s):
--             altpll
--
-- Simulation Library Files(s):
--             altera_mf
-- =====
-- *****
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
-- *****

--Copyright (C) 1991-2010 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors. Please refer to the
--applicable agreement for further details.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
```

Annex

```
USE altera_mf.all;

ENTITY audio_pll IS
  PORT
  (
    inclk0      : IN STD_LOGIC := '0';
    c0          : OUT STD_LOGIC
  );
END audio_pll;

ARCHITECTURE SYN OF audio_pll IS

  SIGNAL sub_wire0      : STD_LOGIC_VECTOR (5 DOWNTO 0);
  SIGNAL sub_wire1      : STD_LOGIC ;
  SIGNAL sub_wire2      : STD_LOGIC ;
  SIGNAL sub_wire3      : STD_LOGIC_VECTOR (1 DOWNTO 0);
  SIGNAL sub_wire4_bv   : BIT_VECTOR (0 DOWNTO 0);
  SIGNAL sub_wire4      : STD_LOGIC_VECTOR (0 DOWNTO 0);

  COMPONENT altpll
  GENERIC (
    clk0_divide_by      : NATURAL;
    clk0_duty_cycle      : NATURAL;
    clk0_multiply_by    : NATURAL;
    clk0_phase_shift    : STRING;
    compensate_clock     : STRING;
    inclk0_input_frequency : NATURAL;
    intended_device_family : STRING;
    lpm_hint             : STRING;
    lpm_type             : STRING;
    operation_mode       : STRING;
    port_activeclock     : STRING;
    port_areset          : STRING;
    port_clkbad0         : STRING;
    port_clkbad1         : STRING;
    port_clkloss         : STRING;
    port_clkswitch       : STRING;
    port_configupdate    : STRING;
    port_fbin            : STRING;
    port_inclk0          : STRING;
    port_inclk1          : STRING;
    port_locked          : STRING;
    port_pfdena         : STRING;
    port_phasecounterselect : STRING;
    port_phasedone       : STRING;
    port_phasestep       : STRING;
    port_phaseupdown     : STRING;
    port_pllena         : STRING;
    port_scanaclr        : STRING;
    port_scanclk         : STRING;
```

Annex

```
port_scanclockena      : STRING;
port_scandata          : STRING;
port_scandataout       : STRING;
port_scandone          : STRING;
port_scanread          : STRING;
port_scanwrite         : STRING;
port_clk0              : STRING;
port_clk1              : STRING;
port_clk2              : STRING;
port_clk3              : STRING;
port_clk4              : STRING;
port_clk5              : STRING;
port_clkena0           : STRING;
port_clkena1           : STRING;
port_clkena2           : STRING;
port_clkena3           : STRING;
port_clkena4           : STRING;
port_clkena5           : STRING;
port_extclk0           : STRING;
port_extclk1           : STRING;
port_extclk2           : STRING;
port_extclk3           : STRING;
);
PORT (
    inclk    : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
    clk      : OUT STD_LOGIC_VECTOR (5 DOWNTO 0)
);
END COMPONENT;

BEGIN
sub_wire4_bv(0 DOWNTO 0) <= "0";
sub_wire4    <= To_stdlogicvector(sub_wire4_bv);
sub_wire1    <= sub_wire0(0);
c0           <= sub_wire1;
sub_wire2    <= inclk0;
sub_wire3    <= sub_wire4(0 DOWNTO 0) & sub_wire2;

altpll_component : altpll
GENERIC MAP (
    clk0_divide_by => 3125,
    clk0_duty_cycle => 50,
    clk0_multiply_by => 1152,
    clk0_phase_shift => "0",
    compensate_clock => "CLK0",
    inclk0_input_frequency => 20000,
    intended_device_family => "Cyclone II",
    lpm_hint => "CBX_MODULE_PREFIX=audio_pll",
    lpm_type => "altpll",
    operation_mode => "NORMAL",
    port_activeclock => "PORT_UNUSED",
    port_areset => "PORT_UNUSED",
    port_clkbad0 => "PORT_UNUSED",
    port_clkbad1 => "PORT_UNUSED",
```

Annex

```
port_clkloss => "PORT_UNUSED",
port_clkswitch => "PORT_UNUSED",
port_configupdate => "PORT_UNUSED",
port_fbin => "PORT_UNUSED",
port_inclk0 => "PORT_USED",
port_inclk1 => "PORT_UNUSED",
port_locked => "PORT_UNUSED",
port_pfdena => "PORT_UNUSED",
port_phasecounterselect => "PORT_UNUSED",
port_phasedone => "PORT_UNUSED",
port_phasestep => "PORT_UNUSED",
port_phaseupdown => "PORT_UNUSED",
port_pllena => "PORT_UNUSED",
port_scanaclr => "PORT_UNUSED",
port_scanclk => "PORT_UNUSED",
port_scanclkena => "PORT_UNUSED",
port_scandata => "PORT_UNUSED",
port_scandataout => "PORT_UNUSED",
port_scandone => "PORT_UNUSED",
port_scanread => "PORT_UNUSED",
port_scanwrite => "PORT_UNUSED",
port_clk0 => "PORT_USED",
port_clk1 => "PORT_UNUSED",
port_clk2 => "PORT_UNUSED",
port_clk3 => "PORT_UNUSED",
port_clk4 => "PORT_UNUSED",
port_clk5 => "PORT_UNUSED",
port_clkena0 => "PORT_UNUSED",
port_clkena1 => "PORT_UNUSED",
port_clkena2 => "PORT_UNUSED",
port_clkena3 => "PORT_UNUSED",
port_clkena4 => "PORT_UNUSED",
port_clkena5 => "PORT_UNUSED",
port_extclk0 => "PORT_UNUSED",
port_extclk1 => "PORT_UNUSED",
port_extclk2 => "PORT_UNUSED",
port_extclk3 => "PORT_UNUSED"
)
PORT MAP (
    inclk => sub_wire3,
    clk => sub_wire0
);

END SYN;

-- =====
-- CNX file retrieval info
-- =====
-- Retrieval info: PRIVATE: ACTIVECLK_CHECK STRING "0"
-- Retrieval info: PRIVATE: BANDWIDTH STRING "1.000"
-- Retrieval info: PRIVATE: BANDWIDTH_FEATURE_ENABLED STRING "0"
```

Annex

```
-- Retrieval info: PRIVATE: BANDWIDTH_FREQ_UNIT STRING "MHz"
-- Retrieval info: PRIVATE: BANDWIDTH_PRESET STRING "Low"
-- Retrieval info: PRIVATE: BANDWIDTH_USE_AUTO STRING "1"
-- Retrieval info: PRIVATE: BANDWIDTH_USE_CUSTOM STRING "0"
-- Retrieval info: PRIVATE: BANDWIDTH_USE_PRESET STRING "0"
-- Retrieval info: PRIVATE: CLKBAD_SWITCHOVER_CHECK STRING "0"
-- Retrieval info: PRIVATE: CLKLOSS_CHECK STRING "0"
-- Retrieval info: PRIVATE: CLKSWITCH_CHECK STRING "1"
-- Retrieval info: PRIVATE: CNX_NO_COMPENSATE_RADIO STRING "0"
-- Retrieval info: PRIVATE: CREATE_CLKBAD_CHECK STRING "0"
-- Retrieval info: PRIVATE: CREATE_INCLK1_CHECK STRING "0"
-- Retrieval info: PRIVATE: CUR_DEDICATED_CLK STRING "c0"
-- Retrieval info: PRIVATE: CUR_FBIN_CLK STRING "e0"
-- Retrieval info: PRIVATE: DEVICE_SPEED_GRADE STRING "Any"
-- Retrieval info: PRIVATE: DIV_FACTOR0 NUMERIC "1"
-- Retrieval info: PRIVATE: DUTY_CYCLE0 STRING "50.00000000"
-- Retrieval info: PRIVATE: EFF_OUTPUT_FREQ_VALUE0 STRING "18.431999"
-- Retrieval info: PRIVATE: EXPLICIT_SWITCHOVER_COUNTER STRING "0"
-- Retrieval info: PRIVATE: EXT_FEEDBACK_RADIO STRING "0"
-- Retrieval info: PRIVATE: GLOCKED_COUNTER_EDIT_CHANGED STRING "1"
-- Retrieval info: PRIVATE: GLOCKED_FEATURE_ENABLED STRING "1"
-- Retrieval info: PRIVATE: GLOCKED_MODE_CHECK STRING "0"
-- Retrieval info: PRIVATE: GLOCK_COUNTER_EDIT NUMERIC "1048575"
-- Retrieval info: PRIVATE: HAS_MANUAL_SWITCHOVER STRING "1"
-- Retrieval info: PRIVATE: INCLK0_FREQ_EDIT STRING "50.000"
-- Retrieval info: PRIVATE: INCLK0_FREQ_UNIT_COMBO STRING "MHz"
-- Retrieval info: PRIVATE: INCLK1_FREQ_EDIT STRING "100.000"
-- Retrieval info: PRIVATE: INCLK1_FREQ_EDIT_CHANGED STRING "1"
-- Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_CHANGED STRING "1"
-- Retrieval info: PRIVATE: INCLK1_FREQ_UNIT_COMBO STRING "MHz"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: PRIVATE: INT_FEEDBACK__MODE_RADIO STRING "1"
-- Retrieval info: PRIVATE: LOCKED_OUTPUT_CHECK STRING "0"
-- Retrieval info: PRIVATE: LONG_SCAN_RADIO STRING "1"
-- Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE STRING "300.000"
-- Retrieval info: PRIVATE: LVDS_MODE_DATA_RATE_DIRTY NUMERIC "0"
-- Retrieval info: PRIVATE: LVDS_PHASE_SHIFT_UNIT0 STRING "deg"
-- Retrieval info: PRIVATE: MIG_DEVICE_SPEED_GRADE STRING "Any"
-- Retrieval info: PRIVATE: MIRROR_CLK0 STRING "0"
-- Retrieval info: PRIVATE: MULT_FACTOR0 NUMERIC "1"
-- Retrieval info: PRIVATE: NORMAL_MODE_RADIO STRING "1"
-- Retrieval info: PRIVATE: OUTPUT_FREQ0 STRING "18.43200000"
-- Retrieval info: PRIVATE: OUTPUT_FREQ_MODE0 STRING "1"
-- Retrieval info: PRIVATE: OUTPUT_FREQ_UNIT0 STRING "MHz"
-- Retrieval info: PRIVATE: PHASE_RECONFIG_FEATURE_ENABLED STRING "0"
-- Retrieval info: PRIVATE: PHASE_RECONFIG_INPUTS_CHECK STRING "0"
-- Retrieval info: PRIVATE: PHASE_SHIFT0 STRING "0.00000000"
-- Retrieval info: PRIVATE: PHASE_SHIFT_STEP_ENABLED_CHECK STRING "0"
-- Retrieval info: PRIVATE: PHASE_SHIFT_UNIT0 STRING "deg"
-- Retrieval info: PRIVATE: PLL_ADVANCED_PARAM_CHECK STRING "0"
-- Retrieval info: PRIVATE: PLL_ARESET_CHECK STRING "0"
-- Retrieval info: PRIVATE: PLL_AUTOPLL_CHECK NUMERIC "1"
-- Retrieval info: PRIVATE: PLL_ENA_CHECK STRING "0"
```


Annex

```
-- Retrieval info: PRIVATE: PLL_ENHPLL_CHECK NUMERIC "0"
-- Retrieval info: PRIVATE: PLL_FASTPLL_CHECK NUMERIC "0"
-- Retrieval info: PRIVATE: PLL_FBMIMIC_CHECK STRING "0"
-- Retrieval info: PRIVATE: PLL_LVDS_PLL_CHECK NUMERIC "0"
-- Retrieval info: PRIVATE: PLL_PFDENA_CHECK STRING "0"
-- Retrieval info: PRIVATE: PLL_TARGET_HARCOPY_CHECK NUMERIC "0"
-- Retrieval info: PRIVATE: PRIMARY_CLK_COMBO STRING "inclk0"
-- Retrieval info: PRIVATE: RECONFIG_FILE STRING "audio_pll.mif"
-- Retrieval info: PRIVATE: SACN_INPUTS_CHECK STRING "0"
-- Retrieval info: PRIVATE: SCAN_FEATURE_ENABLED STRING "0"
-- Retrieval info: PRIVATE: SELF_RESET_LOCK_LOSS STRING "0"
-- Retrieval info: PRIVATE: SHORT_SCAN_RADIO STRING "0"
-- Retrieval info: PRIVATE: SPREAD_FEATURE_ENABLED STRING "0"
-- Retrieval info: PRIVATE: SPREAD_FREQ STRING "50.000"
-- Retrieval info: PRIVATE: SPREAD_FREQ_UNIT STRING "KHz"
-- Retrieval info: PRIVATE: SPREAD_PERCENT STRING "0.500"
-- Retrieval info: PRIVATE: SPREAD_USE STRING "0"
-- Retrieval info: PRIVATE: SRC_SYNCH_COMP_RADIO STRING "0"
-- Retrieval info: PRIVATE: STICKY_CLK0 STRING "1"
-- Retrieval info: PRIVATE: SWITCHOVER_COUNT_EDIT NUMERIC "1"
-- Retrieval info: PRIVATE: SWITCHOVER_FEATURE_ENABLED STRING "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: USE_CLK0 STRING "1"
-- Retrieval info: PRIVATE: USE_CLKENAO STRING "0"
-- Retrieval info: PRIVATE: USE_MIL_SPEED_GRADE NUMERIC "0"
-- Retrieval info: PRIVATE: ZERO_DELAY_RADIO STRING "0"
-- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
-- Retrieval info: CONSTANT: CLK0_DIVIDE_BY NUMERIC "3125"
-- Retrieval info: CONSTANT: CLK0_DUTY_CYCLE NUMERIC "50"
-- Retrieval info: CONSTANT: CLK0_MULTIPLY_BY NUMERIC "1152"
-- Retrieval info: CONSTANT: CLK0_PHASE_SHIFT STRING "0"
-- Retrieval info: CONSTANT: COMPENSATE_CLOCK STRING "CLK0"
-- Retrieval info: CONSTANT: INCLK0_INPUT_FREQUENCY NUMERIC "20000"
-- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "altpll"
-- Retrieval info: CONSTANT: OPERATION_MODE STRING "NORMAL"
-- Retrieval info: CONSTANT: PORT_ACTIVECLOCK STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_ARESET STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_CLKBADO STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_CLKBAD1 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_CLKLOSS STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_CLKSWITCH STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_CONFIGUPDATE STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_FBIN STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_INCLK0 STRING "PORT_USED"
-- Retrieval info: CONSTANT: PORT_INCLK1 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_LOCKED STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_PFDENA STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_PHASECOUNTERSELECT STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_PHASEDONE STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_PHASESTEP STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_PHASEUPDOWN STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_PLENA STRING "PORT_UNUSED"
```

Annex

```
-- Retrieval info: CONSTANT: PORT_SCANACLK STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANCLK STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANCLKENA STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANDATA STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANDATAOUT STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANDONE STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANREAD STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_SCANWRITE STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clk0 STRING "PORT_USED"
-- Retrieval info: CONSTANT: PORT_clk1 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clk2 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clk3 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clk4 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clk5 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clkena0 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clkena1 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clkena2 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clkena3 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clkena4 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_clkena5 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_extclk0 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_extclk1 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_extclk2 STRING "PORT_UNUSED"
-- Retrieval info: CONSTANT: PORT_extclk3 STRING "PORT_UNUSED"
-- Retrieval info: USED_PORT: @clk 0 0 6 0 OUTPUT_CLK_EXT VCC "@clk[5..0]"
-- Retrieval info: USED_PORT: @extclk 0 0 4 0 OUTPUT_CLK_EXT VCC "@extclk[3..0]"
-- Retrieval info: USED_PORT: @inclk 0 0 2 0 INPUT_CLK_EXT VCC "@inclk[1..0]"
-- Retrieval info: USED_PORT: c0 0 0 0 0 OUTPUT_CLK_EXT VCC "c0"
-- Retrieval info: USED_PORT: inclk0 0 0 0 0 INPUT_CLK_EXT GND "inclk0"
-- Retrieval info: CONNECT: @inclk 0 0 1 0 inclk0 0 0 0 0
-- Retrieval info: CONNECT: c0 0 0 0 0 @clk 0 0 1 0
-- Retrieval info: CONNECT: @inclk 0 0 1 1 GND 0 0 0 0
-- Retrieval info: GEN_FILE: TYPE_NORMAL audio_pll.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL audio_pll.ppf TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL audio_pll.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL audio_pll.cmp TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL audio_pll.bsf FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL audio_pll_inst.vhd FALSE
-- Retrieval info: LIB_FILE: altera_mf
-- Retrieval info: CBX_MODULE_PREFIX: ON
```

DATA_ROM.vhd

```
LIBRARY ieee;
    USE ieee.std_logic_1164.all;
    USE ieee.numeric_std.all;

LIBRARY altera_mf;
    USE altera_mf.all;

ENTITY DATA_ROM IS PORT (
    addr          : IN std_logic_vector (12 DOWNTO 0);
    clock         : IN STD_LOGIC ;
    data_from_rom: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
```

Annex

```

END DATA_ROM;

ARCHITECTURE SYN OF DATA_ROM IS
SIGNAL sub_wire0      : STD_LOGIC_VECTOR (15 DOWNT0 0);
COMPONENT altsyncram
  GENERIC (
    clock_enable_input_a      : STRING;
    clock_enable_output_a : STRING;
    init_file                  : STRING;
    intended_device_family    : STRING;
    lpm_hint                   : STRING;
    lpm_type                   : STRING;
    numwords_a                : NATURAL;
    operation_mode            : STRING;
    outdata_aclr_a            : STRING;
    outdata_reg_a            : STRING;
    widthad_a                 : NATURAL;
    width_a                   : NATURAL;
    width_byteena_a          : NATURAL);
  PORT (
    clock0      : IN STD_LOGIC ;
    address_a   : IN STD_LOGIC_VECTOR (12 DOWNT0 0);
    q_a         : OUT STD_LOGIC_VECTOR (15 DOWNT0 0) );
END COMPONENT;

begin

  -- read data from ROM
  data_from_rom <= sub_wire0(15 DOWNT0 0);

  altsyncram_component : altsyncram
    GENERIC MAP (
      clock_enable_input_a => "BYPASS",
      clock_enable_output_a => "BYPASS",
      init_file             => "beebullet.mif",
      intended_device_family => "Cyclone II",
      lpm_hint              =>
"ENABLE_RUNTIME_MOD=NO",
      lpm_type              => "altsyncram",
      numwords_a            => 4680,
      operation_mode        => "ROM",
      outdata_aclr_a        => "NONE",
      outdata_reg_a        => "UNREGISTERED",
      widthad_a             => 13,
      width_a               => 16,
      width_byteena_a      => 1 );
    PORT MAP (
      clock0      => clock,
      address_a   => addr,
      q_a         => sub_wire0 );

End;

```

Annex

Codec_WM8731.vhd

```
-- Audio CODE WM8731
Library ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

Entity Codec_WM8731 is port (
  clk      : in std_logic;          -- Audio CODEC Chip Clock AUD_XCK (18.432 MHz)
  reset_n  : in std_logic;
  disable  : in std_logic;        -- when '1', no output from wm8731
  data_from_rom : in STD_LOGIC_VECTOR(15 downto 0);
  sound_finish : out std_logic_vector(2 downto 0);
  mem_addr  : out unsigned(12 downto 0);
  -- Audio interface signals
  AUD_ADCLRCK : out std_logic;    -- Audio CODEC ADC LR Clock
  AUD_ADCDAT  : in  std_logic;    -- Audio CODEC ADC Data
  AUD_DACLCK  : out std_logic;    -- Audio CODEC DAC LR Clock
  AUD_DACDAT  : out std_logic;    -- Audio CODEC DAC Data
  AUD_BCLK    : inout std_logic;  -- Audio CODEC Bit-Stream Clock
End Codec_WM8731;
```

Architecture RTL of Codec_WM8731 is

```
signal lrck : std_logic;
signal bclk : std_logic;
signal xck  : std_logic;
signal lrck_divider : unsigned(7 downto 0);
signal bclk_divider : unsigned(3 downto 0);
signal set_bclk : std_logic;
signal set_lrck : std_logic;
signal clr_bclk : std_logic;
signal lrck_lat : std_logic;
signal shift_out : STD_LOGIC_VECTOR(15 downto 0);

signal mem_addr_bullet : unsigned(12 downto 0); -- from "bullet" rom to mux
signal counter1 : unsigned(2 downto 0);
signal temp : std_logic;
```

Begin

Process (clk)

```
begin
  if rising_edge(clk) then
    if reset_n = '0' then
      lrck_divider <= (others => '0');
    elsif lrck_divider = X"BF" then -- X"BF" = (X"C0" - X"1") : lrck_divider = (18.432
MHz/48 KHz)-1
      lrck_divider <= X"00";
    else
      lrck_divider <= lrck_divider + 1;
    end if;
  end if;
End process;
```

Annex

```
Process (clk)
begin
    if rising_edge(clk) then
        if reset_n = '0' then
            bclk_divider <= (others => '0');
        elsif bclk_divider = X"B" or set_lrck = '1' then
            bclk_divider <= X"0";
        else
            bclk_divider <= bclk_divider + 1;
        end if;
    end if;
End process;

set_lrck <= '1' when lrck_divider = X"BF" else '0';

Process (clk)
begin
    if rising_edge(clk) then
        if reset_n = '0' then
            lrck <= '0';
        elsif set_lrck = '1' then
            lrck <= not lrck;
        end if;
    end if;
End process;

-- BCLK divider
set_bclk <= '1' when bclk_divider(3 downto 0) = "0101" else '0'; -- X"5"
clr_bclk <= '1' when bclk_divider(3 downto 0) = "1011" else '0'; -- X"B"

Process (clk)
begin
    if rising_edge(clk) then
        if reset_n = '0' then
            bclk <= '0';
        elsif set_lrck = '1' or clr_bclk = '1' then
            bclk <= '0';
        elsif set_bclk = '1' then
            bclk <= '1';
        end if;
    end if;
End process;

-- Audio data shift output
Process (clk)
begin
    if rising_edge(clk) then
        if reset_n = '0' then
            shift_out <= (others => '0');
        elsif set_lrck = '1' then
            shift_out <= data_from_rom;
        elsif clr_bclk = '1' then
            shift_out <= shift_out (14 downto 0) & '0';
        end if;
    end if;
End process;
```

Annex

```
        end if;
        -- when disable = 1, no audio data output, which means mute.
        if disable = '1' then
            temp <= '0';
        else
            temp <= shift_out(15);
        end if;
    end if;
End process;

-- Audio outputs
AUD_ADCLRCK <= lrck;
AUD_DACLCK <= lrck;
AUD_BCLK <= bclk;
AUD_DACDAT <= temp;

-- Counter 1 for ROM01
Process(clk)
begin
    if rising_edge(clk) then
        if reset_n = '0' then
            mem_addr_bullet <= (others => '0');
            sound_finish(0) <= '0';
            counter1 <= "000";
        elsif lrck_lat = '1' and lrck = '0' then
            if counter1 = "101" then          -- counter1 = X"5"
                counter1 <= "000";
                if mem_addr_bullet = x"0dff" then -- End addr of memery
                    mem_addr_bullet <= (others => '0');
                    sound_finish(0) <='1';
                else
                    mem_addr_bullet <= mem_addr_bullet + 1;
                end if;
            else
                counter1 <= counter1 + 1;
            end if;
        end if;
    end if;
    mem_addr <= mem_addr_bullet;
End process;

-- LRCK divider
-- Audio chip main clock is 18.432MHz / Sample rate 48KHz
-- Divider is 18.432 MHz / 48KHz / 2 = 192 (X"C0")
-- Left justify mode set by I2C controller
-----
Process(clk)
begin
    if rising_edge(clk) then
        lrck_lat <= lrck;
    End if;
End process;
```

Annex

End architecture;

I2C_sound.v

```

module I2C_sound
    (
        CLOCK_50,           // 50 MHz
        KEY,                 // Pushbutton
        I2C_SDAT,           // I2C Data
        I2C_SCLK,           // I2C Clock
        r
    );

input    CLOCK_50;
input    KEY;
inout    I2C_SDAT;
output   I2C_SCLK;
output   [6:0] r;           // led r

assign r = VOL;

I2C_AV_Config u1 ( // Host Side
                  .iCLK(CLOCK_50),
                  .iRST_N(KEY),
                  .iVOL(VOL),
                  // I2C Side
                  .I2C_SCLK(I2C_SCLK),
                  .I2C_SDAT(I2C_SDAT) );

reg [6:0] VOL;

always@(negedge KEY)
begin
    if( VOL < 68 )
        VOL <= 98 ;
    else
        VOL <= VOL + 3;
end
endmodule

```

I2C_AV_Config.v

```

module I2C_AV_Config (
// Host Side
    iCLK,
    iRST_N,
    iVOL,
// I2C Side
    I2C_SCLK,
    I2C_SDAT );

// Host Side
input    iCLK;
input    iRST_N;
input    [6:0] iVOL;

```

Annex

```

//      I2C Side
output      I2C_SCLK;
inout       I2C_SDAT;

//      Internal Registers/Wires
reg [15:0]  mI2C_CLK_DIV;
reg [23:0]  mI2C_DATA;
reg                mI2C_CTRL_CLK;
reg                mI2C_GO;
wire          mI2C_END;
wire          mI2C_ACK;
reg [15:0]  LUT_DATA;
reg [3:0]   LUT_INDEX;
reg [1:0]   mSetup_ST;

//      Clock Setting
parameter CLK_Freq      = 50000000; // 50 MHz
parameter I2C_Freq      = 20000;    // 20 KHz
//      LUT Data Number
parameter LUT_SIZE      = 2;
//      Audio Data Index
//parameter Dummy_DATA      = 0;
//parameter REG_RESET      = 1;
//parameter REG_LLINEIN    = 2;
//parameter REG_RLINEIN    = 3;

parameter REG_LHPOUT      = 0;
parameter REG_RHPOUT      = 1;

//parameter REG_ANA_PATH_CTRL = 6;
//parameter REG_DIG_PATH_CTRL = 7;
//parameter REG_POWER_ON     = 8;
//parameter REG_FORMAT       = 9;
//parameter REG_SAMPLE_CTRL  = 10;
//parameter REG_ACTIVE_CTRL  = 11;

////////// I2C Control Clock //////////
always@(posedge iCLK or negedge iRST_N)
begin
    if(!iRST_N)
    begin
        mI2C_CTRL_CLK    <= 0;
        mI2C_CLK_DIV <= 0;
    end
    else
    begin
        if( mI2C_CLK_DIV < (CLK_Freq/I2C_Freq) )
            mI2C_CLK_DIV <= mI2C_CLK_DIV + 1;
        else
        begin
            mI2C_CLK_DIV <= 0;
            mI2C_CTRL_CLK <= ~mI2C_CTRL_CLK;
        end
    end
end

```


Annex

```

end
end
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
I2C_Controller u0 ( .CLOCK(mi2C_CTRL_CLK), // Controller Work Clock
                   .I2C_SCLK(I2C_SCLK), // I2C
                   .I2C_SDAT(I2C_SDAT), //
                   .I2C_DATA(mi2C_DATA), //
                   DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
                   .GO(mi2C_GO), //
                   .END(mi2C_END),
                   // END transfor
                   .ACK(mi2C_ACK),
                   // ACK
                   .RESET(iRST_N) );

////////////////////////////////////////////////////////////////// Config Control ////////////////////////////////////////////////////////////////////
always@(posedge mi2C_CTRL_CLK or negedge iRST_N)
begin
    if(!iRST_N)
    begin
        LUT_INDEX <= 0;
        mSetup_ST <= 0;
        mi2C_GO <= 0;
    end
    else
    begin
        if(LUT_INDEX < LUT_SIZE)
        begin
            case(mSetup_ST)
            0: begin
                mi2C_DATA <= {8'h34,LUT_DATA}; //I2C ADDRESS Chip
                mi2C_GO <= 1;
                mSetup_ST <= 1;
            end
            1: begin
                if(mi2C_END)
                begin
                    if(!mi2C_ACK)
                    mSetup_ST <= 2;
                    else
                    mSetup_ST <= 0;
                    mi2C_GO <= 0;
                end
            end
            2: begin
                LUT_INDEX <= LUT_INDEX+1;
                mSetup_ST <= 0;
            end
        end
    end
end

```

Annex

```

                                endcase
                            end
                        end
                    end
                end
            end
        end
    end
end
////////////////////// Config Data LUT ////////////////////////
always
begin
    case(LUT_INDEX)
        // Audio Configuration Register Data
        // Dummy_DATA : LUT_DATA <= 16'h0000;
        // REG_RESET : LUT_DATA <= 16'h1E00;
        // REG_LLINEIN : LUT_DATA <= 16'h001A;
        // REG_RLINEIN : LUT_DATA <= 16'h021A;
        // REG_LHPOUT : LUT_DATA <= {7'h02,1'b0,1'b0,iVOL};
        // REG_RHPOUT : LUT_DATA <= {7'h03,1'b0,1'b0,iVOL};
        // REG_ANA_PATH_CTRL : LUT_DATA <= 16'h08F8;
        // REG_DIG_PATH_CTRL : LUT_DATA <= 16'h0A06;
        // REG_POWER_ON : LUT_DATA <= 16'h0C00;
        // REG_FORMAT : LUT_DATA <= 16'h0E41;
        // REG_SAMPLE_CTRL : LUT_DATA <= 16'h1002;
        // REG_ACTIVE_CTRL : LUT_DATA <= 16'h1201;
        // default : LUT_DATA <= 16'h0000;
    endcase
end
endmodule
```

Annexe B
Caractéristique du WM8731

La **Figure 1** et le **Tableau 2** illustré les caractéristiques du codec audio WM8731.

Tableau 1:

Description	symp	value	Unit
TEMPERATURE RANGE		-40 to +85	°C
VDD RANGE		1.8 to 3.6	V
sampling rates	clk	8.0 32.0 44.1 48.0 88.2 96.0	KHz
Signal to Noise Ratio	SNR	85	dB
Dynamic Range	DR	85	dB
Total Harmonic Distortion	THD	-84	dB
Power Supply Rejection Ratio	PSRR	50	1kHz, 100mVpp
Microphone input	volume	-6 to +34	dB
	Current	1.5	mA
Line inputs	volume	+12 to -34	dB
	Current	0.7	mA
headphone output	volume	+6 to -73	dB

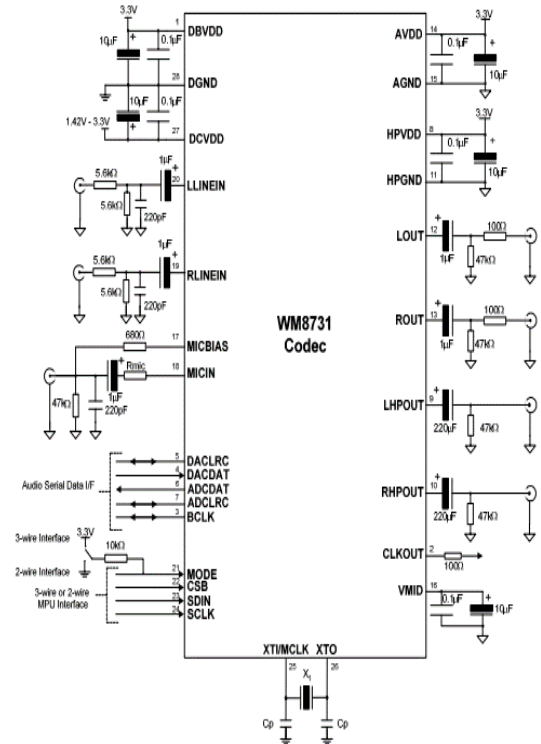


Figure 1: Diagramme des composant externe.