



الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Akli Mohand Oulhadj - Bouira -
Tasdawit Akli Muḥend Ulḥağ - Tubirett -



وزارة التعليم العالي والبحث العلمي
جامعة أكلي محمد أولحاج
- البويرة -

Mémoire de Master

Présenté au

Département : Génie Électrique

Domaine : Sciences et Technologies

Filière : Electronique

Spécialité : Electronique des systèmes embarqués

Réalisé par :

Merzouk Lounas

Thème

**Implémentation d'une commande PID sur FPGA
pour le contrôle d'un moteur électrique**

Soutenu le: **23/06/2025**

Devant la commission composée de :

Mr. Moudache.S		Univ. Bouira	Président
Mr. HAROUN Smail	M.C.B	Univ. Bouira	Encadreur
Mr. Chelbi.S		Univ. Bouira	Examineur

DEDICACES

JE DÉDIE CE MODESTE TRAVAIL À ***MA FEMME.***

MON FILS, LE ***GÉNIE, ILYANE***

MES DEUX ***PRINCESSES ARINES*** ET ***ALICIA***

JE DÉDIE CE TRAVAIL À MA MÈRE ET MON PÈRE.

REMERCIEMENTS

***JE TIENS À REMÉRCIER MON
ENCADREUR MONSIEUR HAROUN EN
PARTICULIER POUR SON
ACCOMPAGNEMENT TOUT AU LONG
DE MON CURSUS.***

***JE TIENS À REMÉRICIER TOUT NOS
PROFFESSEURS CHACUN SON NOM ET
TOUT LE PERSONNEL DE NOTRE
DÉPARTEMENT GÉNIE ÉLECTRIQUE.***

***JE TIENS À REMÉRCIER TOUT CEUX
QUI ONT CONTRIBUÉ DE PRET ET DE
LOIN À LA RÉALISATION DE CE
MODESTE TRAVAIL DU MONDE
NUMÉRIQUE.***

RESUMÉ :

Ce mémoire présente l'implémentation et la validation expérimentale d'un système de régulation de vitesse par correcteur PID numérique, implanté sur circuit FPGA Gowin GWINSR-4C (carte Tang Nano 4K) en utilisant le langage VHDL. Le procédé commandé est un motoréducteur à courant continu équipé d'un encodeur incrémental. Le réglage du correcteur PI est réalisé à l'aide de l'outil MATLAB PID Tuner, puis affiné par itérations expérimentales sur le système réel. La validation expérimentale démontre une erreur statique nulle sur la plage de consignes. La mesure stroboscopique valide le système à une erreur de 0,6 %. La liaison FPGA et le cœur ARM Cortex-M3 embarqué est assuré par la liaison UART interne.

Mots-clés : PID numérique, FPGA, VHDL, Moteur CC, Identification expérimentale, UART, Gowin GWINSR-4C, Tang Nano 4K, ARM Cortex-M3.

ABSTRACT (ENGLISH):

This thesis presents the implementation and experimental validation of a speed regulation system using a digital PID controller, implemented on a Gowin GWINSR-4C FPGA chip (Tang Nano 4K board) using the VHDL language. The controlled plant is a DC gear motor equipped with an incremental encoder. The PI controller tuning is performed using the MATLAB PID Tuner tool, then refined through experimental iterations on the real system. Experimental validation demonstrates zero steady-state error over the setpoint range. Stroboscopic measurement validates the system with an error of 0.6%. Communication between the FPGA and the embedded ARM Cortex-M3 core is ensured via the internal UART link.

Keywords: Digital PID, FPGA, VHDL, DC Motor, Experimental Identification, UART, Gowin GWINSR-4C, Tang Nano 4K, ARM Cortex-M3.

LISTE DES ABRÉVIATIONS :

<i>API</i>	<i>Automate Programmable Industriel (Programmable Logic Controller, PLC)</i>
<i>ASCII</i>	<i>American Standard Code for Information Interchange</i>
<i>BF</i>	<i>Boucle Fermée</i>
<i>BO</i>	<i>Boucle Ouverte</i>
<i>BRAM</i>	<i>Block Random Access Memory</i>
<i>BSRAM</i>	<i>Block Static Random Access Memory</i>
<i>CC</i>	<i>Courant Continu</i>
<i>CPU</i>	<i>Central Processing Unit</i>
<i>CSP</i>	<i>Chip Scale Package</i>
<i>D</i>	<i>Dérivé (action dérivée du régulateur PID)</i>
<i>DC</i>	<i>Direct Current (Courant Continu)</i>
<i>DSP</i>	<i>Digital Signal Processor (Processeur de Signal Numérique)</i>
<i>DSP18</i>	<i>Multiplicateur câblé 18 × 18 bits (Gowin)</i>
<i>EDA</i>	<i>Electronic Design Automation</i>
<i>E/S</i>	<i>Entrées/Sorties</i>
<i>FEM</i>	<i>Force Électromotrice</i>
<i>FF</i>	<i>Flip-Flop (Bascule D)</i>
<i>FPGA</i>	<i>Field-Programmable Gate Array (Réseau de Portes Programmables sur Site)</i>
<i>f.e.m.</i>	<i>Force Électromotrice</i>
<i>Fmax</i>	<i>Fréquence Maximale de fonctionnement</i>
<i>G(p)</i>	<i>Fonction de Transfert du Procédé (domaine de Laplace)</i>
<i>G(s)</i>	<i>Fonction de Transfert du Procédé (domaine de Laplace)</i>
<i>Gowin</i>	<i>Gowin Semiconductor (fabricant FPGA)</i>
<i>GPIO</i>	<i>General Purpose Input/Output (Entrée/Sortie à Usage Général)</i>
<i>GPC</i>	<i>Generalized Predictive Control (Commande Prédictive Généralisée)</i>
<i>H(p)</i>	<i>Fonction de Transfert du Capteur (domaine de Laplace)</i>
<i>HDL</i>	<i>Hardware Description Language (Langage de Description Matérielle)</i>
<i>I</i>	<i>Intégral (action intégrale du régulateur PID)</i>
<i>I/O</i>	<i>Input/Output (Entrée/Sortie)</i>
<i>I²C</i>	<i>Inter-Integrated Circuit (bus série synchrone multimaster)</i>
<i>IP</i>	<i>Intellectual Property (Propriété Intellectuelle — bloc de conception réutilisable)</i>
<i>JGA</i>	<i>Série de motoréducteurs (JGA25-370)</i>
<i>JTAG</i>	<i>Joint Test Action Group (interface de test et programmation)</i>
<i>Kc</i>	<i>Constante de Couple (N·m/A)</i>
<i>Ke</i>	<i>Constante de Force Électromotrice (V·s/rad)</i>
<i>Ki</i>	<i>Gain Intégral du régulateur PID</i>
<i>Km</i>	<i>Gain Statique du moteur (rad·s⁻¹·V⁻¹)</i>
<i>Kp</i>	<i>Gain Proportionnel du régulateur PID</i>
<i>Kd</i>	<i>Gain Dérivé du régulateur PID</i>
<i>L</i>	<i>Inductance (H)</i>
<i>LED</i>	<i>Light-Emitting Diode (Diode Électroluminescente)</i>
<i>LF</i>	<i>Line Feed (Saut de Ligne, caractère ASCII 0x0A)</i>
<i>LSB</i>	<i>Least Significant Bit (Bit de Poids Faible)</i>
<i>LUT</i>	<i>Look-Up Table (Table de Consultation — élément logique FPGA)</i>
<i>LUT4</i>	<i>Look-Up Table à 4 entrées</i>
<i>LVC MOS</i>	<i>Low-Voltage Complementary Metal-Oxide-Semiconductor</i>
<i>M3</i>	<i>Filetage métrique de diamètre 3 mm</i>
<i>MCU</i>	<i>Microcontroller Unit (Microcontrôleur)</i>
<i>MISO</i>	<i>Master In Slave Out (SPI)</i>

MOSI *Master Out Slave In (SPI)*
 MSE *Master of Science in Engineering (Master en Sciences de l'Ingénieur)*
 MSB *Most Significant Bit (Bit de Poids Fort)*
 N *Coefficient de filtrage de l'action dérivée*
 Nenc *Nombre d'impulsions par tour de l'encodeur (PPR)*
 Nimp *Nombre d'impulsions comptées*
 P&R *Placement et Routage (Place and Route)*
 PCB *Printed Circuit Board (Carte de Circuit Imprimé)*
 PID *Proportionnel-Intégral-Dérivé (régulateur)*
 PLL *Phase-Locked Loop (Boucle à Verrouillage de Phase)*
 PPR *Pulses Per Revolution (Impulsions par Tour)*
 PWM *Pulse Width Modulation (Modulation de Largeur d'Impulsion)*
 Q15 *Format de représentation en virgule fixe sur 16 bits (1 signe + 15 fractionnaires)*
 QFN *Quad Flat No-leads (boîtier plat sans pattes)*
 RAM *Random Access Memory (Mémoire à Accès Aléatoire)*
 R² *Coefficient de Détermination (indicateur statistique d'ajustement)*
 ROM *Read-Only Memory (Mémoire Morte)*
 RPM *Revolutions Per Minute (Tours par Minute)*
 RTOS *Real-Time Operating System (Système d'Exploitation Temps Réel)*
 RX *Reception (Réception)*
 SCL *Serial Clock (horloge I²C)*
 SDA *Serial Data (données I²C)*
 SoC *System on Chip (Système sur Puce)*
 SPI *Serial Peripheral Interface (bus série synchrone)*
 SRAM *Static Random Access Memory*
 STD *Standard*
 T *Constante de Temps (s)*
 Td *Temps Dérivé (s)*
 Te *Période d'Échantillonnage (s)*
 Ti *Temps Intégral (s)*
 TO-220 *Transistor Outline 220 (boîtier de régulateur de tension)*
 tr *Temps de Montée (rise time)*
 tr5 *Temps de Réponse à ±5 %*
 TX *Transmission*
 UART *Universal Asynchronous Receiver-Transmitter (Émetteur-Récepteur Asynchrone Universel)*
 USB *Universal Serial Bus (Bus Série Universel)*
 VCC *Voltage at Common Collector (tension d'alimentation positive)*
 VHDL *VHSIC Hardware Description Language*
 VHSIC *Very High Speed Integrated Circuit*
 ω *Vitesse Angulaire (rad/s)*
 ω_{ref} *Vitesse de Référence (consigne de vitesse)*
 Z *Variable de la Transformée en Z (domaine discret)*

SYMBOLES ET NOTATIONS MATHÉMATIQUES

$\int$	<i>Intégrale</i>
Σ	<i>Sommation (sigma)</i>
Δ	<i>Incrément (delta)</i>
α	<i>Rapport Cyclique (duty cycle)</i>
ε	<i>Erreur (epsilon)</i>
ε_s	<i>Erreur Statique</i>
τ	<i>Constante de Temps (tau)</i>
τ_e	<i>Constante de Temps Électrique</i>
τ_m	<i>Constante de Temps Mécanique</i>
ϕ_m	<i>Marge de Phase (phi)</i>
ω_{BF}	<i>Bande Passante de la Boucle Fermée</i>

LES UNITÉS

A	<i>Ampère (courant électrique)</i>
bit	<i>Binary digit (chiffre binaire)</i>
$Baud$	<i>Unité de débit de symboles (symboles/seconde)</i>
dB	<i>Décibel (rapport logarithmique)</i>
Hz	<i>Hertz (fréquence)</i>
$kg \cdot m^2$	<i>Kilogramme-mètre carré (moment d'inertie)</i>
mH	<i>Millihenry (inductance)</i>
ms	<i>Milliseconde</i>
$N \cdot m$	<i>Newton-mètre (couple)</i>
$N \cdot m \cdot s / rad$	<i>Coefficient de frottement visqueux</i>
Ω	<i>Ohm (résistance électrique)</i>
rad	<i>Radian (angle)</i>
rad/s	<i>Radian par seconde (vitesse angulaire)</i>
s	<i>Seconde (temps)</i>
Tr/min	<i>Tours par minute (vitesse de rotation)</i>
V	<i>Volt (tension électrique)</i>
$V \cdot s / rad$	<i>Constante de force électromotrice</i>
W	<i>Watt (puissance)</i>
μs	<i>Microseconde</i>

Liste des Tableaux

Chapitre I : Asservissement, régulation PID et implémentation numérique

Tableau I.1 — Influence qualitative de chaque action PID sur les critères de performance ...	20
Tableau I.2 — Comparaison des technologies d'implémentation du régulateur PID	23
Tableau I.3 — Comparaison des méthodes de discrétisation du régulateur PID.....	24

Chapitre II : Présentation des matériels et logiciels utilisés

Tableau II.1 — Ressources du circuit intégré GW1NSR-4C exploitées dans le projet.....	30
Tableau II.2 — Caractéristiques du motoréducteur JGA25-3702.....	32
Tableau II.3 — Paramètres de configuration du L298N dans le projet	34
Tableau II.4 — Bilan matériel complet de la chaîne de commande.....	41

Chapitre III : Modélisation du moteur et réglage du correcteur PI

Tableau III.1 — Paramètres du modèle identifié en boucle ouverte.....	48
Tableau III.2 — Démonstration formelle du changement d'échelle	53
Tableau III.3 — Correspondance entre paramètres MATLAB et constantes VHDL.....	54
Tableau III.4 — Résultats de validation en boucle fermée sur la plage opérationnelle.....	54

Chapitre IV : Implémentation sur FPGA et validation expérimentale

Tableau IV.1 — Affectation des broches du FPGA (configuration SoC finale).....	58
Tableau IV.2 — Procédure de validation du décodage de l'encodeur	61
Tableau IV.3 — Calibration itérative du gain proportionnel KP.....	63
Tableau IV.4 — Corrections apportées au firmware MCU lors de l'intégration	66
Tableau IV.5 — Ressources FPGA : module PID seul (Phase 4 — boucle fermée).....	68
Tableau IV.6 — Ressources FPGA : configuration SoC complète (Phase 5).....	68
Tableau IV.7 — Résultats de validation en boucle fermée sur la plage opérationnelle.....	69
Tableau IV.8 — Limites identifiées et pistes d'amélioration.....	71
Tableau F.1 — Ressources et caractéristiques du circuit intégré GW1NSR.....	86
Tableau F.2 — Caractéristiques expérimentales du motoréducteur JGA25-370.....	86
Tableau F.3 — Paramètres de configuration du pont en H L298N dans le projet.....	87
Tableau F.4 — Caractéristiques de l'afficheur OLED SSD1306.....	88

Liste des Figures

Chapitre I : Asservissement, régulation PID et implémentation numérique

Figure I.1 — Schéma fonctionnel d'un système asservi en boucle fermée.....	17
Figure I.2 — Organigramme de l'algorithme du PID numérique discret.....	26

Chapitre II : Présentation des matériels et logiciels utilisés

Figure II.1 — Carte de développement Tang Nano 4K.....	30
Figure II.2 — Interface USB-JTAG intégrée et brochage de la carte Tang Nano 4K.....	31
Figure II.3 — Motoréducteur JGA25-370.....	33
Figure II.4 — Module pont en H L298N et schéma de câblage avec le FPGA.....	34
Figure II.5 — Analyseur logique à 8 entrées.....	35
Figure II.6 — Terminal PuTTY	36
Figure II.7 — Écran OLED SSD1306 de résolution 128×64 pixels.....	36
Figure II.8 — Arduino Uno utilisé comme relais UART → I2C vers l'écran OLED.....	37
Figure II.9 — Schéma complet de la chaîne d'affichage FPGA → Arduino → OLED.....	37
Figure II.10 — Interface GowinEDA : synthèse et rapport d'utilisation des ressources.....	38
Figure II.11 — Environnement de développement Eclipse / Gowin MCU SDK.....	39
Figure II.12 — Interface MATLAB PID Tuner	40
Figure II.13 — interface du logiciel PulseView.....	40
Figure II.14 — interface logiciel du terminal PuTTY.....	41
Figure II.15 — Photo du banc d'essai complet	42

Chapitre III : Modélisation du moteur et réglage du correcteur PI

Figure III.1 — Schéma synoptique du système à commander.....	44
Figure III.2 — Signaux enregistrés par l'analyseur logique (test en boucle ouverte).....	48
Figure III.3 — Réponse indicielle : mesure expérimentale et modèle identifié ($R^2 = 0,947$)...49	
Figure III.4 — Réglage du correcteur PI dans l'interface MATLAB PID Tuner.....	51
Figure III.5 — Réponse indicielle en boucle fermée simulée sous MATLAB ($T_e = 10$ ms)....52	
Figure III.6 — Validation en boucle fermée à l'analyseur logique ($K_P = 13$, $K_I = 24$).....	55

Chapitre IV : Implémentation sur FPGA et validation expérimentale

Figure IV.1 — Schéma synoptique de l'architecture complète du système.....	58
Figure IV.2 — Signal PWM généré par le FPGA, mesuré à l'analyseur logique.....	59
Figure IV.3 — Signaux en quadrature A et B de l'encodeur incrémental.....	60
Figure IV.4 — Comptage de 10 impulsions par fenêtre temporelle de 10 ms.....	61
Figure IV.5 — Trame UART capturée à l'analyseur logique en régime permanent.....	65
Figure IV.6 — Affichage de la trame UART sur le terminal PuTTY.....	66
Figure IV.7 — État haut des lignes SCL et SDA au reset (pull-up).....	67
Figure IV.8 — État bas lors de l'activation du protocole I2C.....	67
Figure IV.9 — la vérification du fonctionnement et du non-blocage MCU.....	67
Figure IV.10 — Affichage OLED via Arduino Uno (relaisUART)UART.....	67
Figure IV.11 — Mesure stroboscopique de la vitesse réelle (consigne = 5 imp./10 ms).....	70
Figure IV.12 — Cohérence entre simulation MATLAB et mesure expérimentale.....	70
Figure A.1__ Architecture câblé de la phase 4.....	80
Figure A.2__ Architecture câblé de la phase 5.....	81
Figure G.1 — Résultats de synthèse GowinEDA(Phase 4)	89
Figure G.2 — Résultats de synthèse GowinEDA(Phase 5).....	90

Table des matières

Dédicace	i
Remerciements	ii
Résumé (français).....	iii
Abstract (anglais)	iii
Liste des abréviations	iv
Liste des tableaux	vii
Liste des figures	viii
Table des matieres.....	ix
Introduction générale.....	xiv

CHAPITRE I

Asservissement, régulation PID et implémentation numérique

Introduction	15
I.1 L'Asservissement en Boucle Fermée	15
<i>I.1.1 Définition et principe</i>	<i>15</i>
<i>I.1.2 Schéma fonctionnel d'un système asservi</i>	<i>16</i>
<i>I.1.3 Critères de performance d'un asservissement</i>	<i>17</i>
I.2 Le Régulateur PID	18
<i>I.2.1 Structure générale et équation de commande.....</i>	<i>18</i>
<i>I.2.2 L'action Proportionnelle.....</i>	<i>18</i>
<i>I.2.3 L'action Intégrale</i>	<i>18</i>
<i>I.2.4 L'action Dérivée</i>	<i>19</i>
<i>I.2.5 Effets combinés — Synthèse</i>	<i>19</i>
<i>I.2.6 Structures d'association des actions.....</i>	<i>20</i>
<i>I.2.7 Limites du régulateur PID.....</i>	<i>20</i>
I.3 Méthodes de Réglage du Régulateur PID	20
<i>I.3.1 Méthode de Ziegler-Nichols</i>	<i>20</i>
<i>I.3.2 Méthode de placement de pôles</i>	<i>21</i>
<i>I.3.3 Optimisation par simulation</i>	<i>21</i>
I.4 Technologies d'Implémentation Embarquée	21
<i>I.4.1 Implémentation analogique.....</i>	<i>21</i>
<i>I.4.2 Implémentation sur automate programmable (API)</i>	<i>21</i>
<i>I.4.3 Implémentation sur microcontrôleur ou DSP</i>	<i>22</i>
<i>I.4.4 Implémentation sur FPGA</i>	<i>22</i>
<i>I.4.5 Comparaison des technologies</i>	<i>22</i>
I.5 Discrétisation du Régulateur PID	23
<i>I.5.1 Nécessité du passage au domaine discret.....</i>	<i>23</i>

I.5.2 Méthodes de discrétisation.....	23
I.5.3 Forme position et forme récurrente	24
I.5.4 Algorithme complet du PID numérique.....	24
I.6 Représentation en Virgule Fixe	25
I.6.1 Principe du format virgule fixe	25
I.6.2 Conversion des paramètres PID et facteur d'échelle SCALE	26
Conclusion.....	26
II.1 Introduction	28
II.2 Plateforme FPGA : Tang Nano 4K (GW1NSR-4C).....	28
II.2.1 Architecture du circuit intégré.....	28
II.2.2 Ressources logiques et caractéristiques électriques	29
II.2.3 Programmation et débogage	30
II.3 Actionneur : Motoréducteur JGA25-370	30
II.3.1 Description et caractéristiques	30
II.3.2 Rôle de l'encodeur dans la boucle de régulation	31
II.3.3 Adaptation des niveaux de tension	31
II.4 Interface de Puissance : Pont en H L298N.....	32
II.4.1 Principe de fonctionnement	32
II.4.2 Caractéristiques électriques et contraintes.....	32
II.4.3 Protection contre les inversions de sens : temps mort	33
II.5 Instruments de Mesure et de Débogage	34
II.5.1 Analyseur logique PulseView	34
II.5.2 Multimètre	34
II.5.3 Terminal PuTTY	34
II.6 Affichage : Écran OLED SSD1306 et Arduino Uno	35
II.6.1 Écran OLED SSD1306 — Caractéristiques et contrainte de tension	35
II.6.2 Arduino Uno — Rôle de convertisseur de protocole	35
II.6.3 Convertisseur de niveau logique 1,8 V → 5 V	36
II.7 Environnements Logiciels	36
II.7.1 GowinEDA.....	36
II.7.2 Gowin MCU SDK (Eclipse)	37
II.7.3 MATLAB et MATLAB PID Tuner	38
II.7.4 PulseView	39
II.7.5 PuTTY.....	39
II.8 Récapitulatif de la Chaîne Matérielle Complète	40
II.9 Conclusion.....	41
Introduction	42

III.1 Présentation du Système Étudié	42
III.2 Méthodologie de Modélisation	43
III.2.1 Choix de l'approche d'identification	43
III.2.2 Hypothèse du modèle du premier ordre	44
III.2.3 Justification du signal d'excitation	44
III.3 Protocole Expérimental d'Identification en Boucle Ouverte	44
III.3.1 Description du banc d'essai	44
III.3.2 Mise en mode boucle ouverte et procédure d'acquisition	45
III.4 Traitement des Données et Identification du Modèle	45
III.4.1 Pipeline de traitement MATLAB	45
III.4.2 Résultats de l'identification	46
Les résultats obtenus sont regroupés dans le Tableau III.1.....	46
III.5 Difficultés Rencontrées et Solutions Apportées	48
III.5.1 Bruit de quantification à basse vitesse	48
III.5.2 Zone morte à très basse vitesse	48
III.5.3 Saturation du pont en H L298N	49
III.6 Synthèse du Correcteur PI avec MATLAB PID Tuner	49
III.6.1 Justification de la structure PI	49
III.6.2 Pourquoi MATLAB PID Tuner plutôt que Ziegler-Nichols ?	49
III.6.3 Déroulement du réglage	49
III.6.4 Paramètres du correcteur obtenus	50
III.7 Validation par Simulation de la Boucle Fermée	50
III.7.1 Configuration de la simulation	50
III.7.2 Résultats et analyse	51
III.8 Démonstration Formelle du Changement d'Échelle MATLAB → VHDL	52
III.8.1 Formulation du problème	52
III.8.2 Démonstration du facteur d'échelle	52
III.8.3 Implantation arithmétique et gestion des débordements	53
III.9 Validation Expérimentale en Boucle Fermée	54
Conclusion	55
Introduction	56
IV.1 Description de l'Architecture Matérielle	56
IV.1.1 Schéma fonctionnel du système	56
IV.1.2 Affectation des broches	57
IV.2 Génération du Signal PWM et Contrôle du Moteur	58
IV.2.1 Principe de la modulation de largeur d'impulsion	58
IV.2.2 Validation de la génération du signal PWM	58

IV.3 Implémentation de l'Encodeur sur FPGA	59
<i>IV.3.1 Décodage des signaux en quadrature</i>	59
<i>IV.3.2 Comptage de vitesse sur fenêtre temporelle</i>	59
<i>IV.3.3 Validation du décodage encodeur</i>	60
IV.4 Implémentation du Correcteur PI sur FPGA	60
<i>IV.4.1 Structure du correcteur discret et implantation arithmétique</i>	60
<i>IV.4.2 Anti-windup et saturation de la commande</i>	61
<i>IV.4.3 Calibration itérative des paramètres</i>	62
<i>IV.4.4 Découverte et analyse de l'instabilité en haute consigne</i>	62
IV.5 Transition vers l'Affichage OLED et Pivot Architectural	63
<i>IV.5.1 Impasse de l'interface FPGA-Arduino directe</i>	63
<i>IV.5.2 Pivot vers le microcontrôleur embarqué</i>	63
IV.6 Intégration SoC : FPGA, MCU, UART et OLED	64
<i>IV.6.1 Architecture du système sur puce</i>	64
<i>IV.6.2 Validation du protocole UART</i>	64
<i>IV.6.3 validation du firmware MCU</i>	65
<i>IV.6.4 affichage sur OLED</i>	66
<i>IV.6.5 Occupation des ressources FPGA</i>	67
IV.7 Validation Finale et Cohérence avec la Simulation	68
<i>IV.7.1 Validation des paramètres PID sur la plage opérationnelle</i>	68
<i>IV.7.2 Vérification stroboscopique de la vitesse</i>	68
<i>IV.7.3 Cohérence avec la simulation MATLAB</i>	69
IV.8 Perspectives et Limites du Système	70
Conclusion	70
CONCLUSION GÉNÉRALE	72
Références Bibliographiques	74
ANNEXES	76
Annexe A — architecture câblé du projet généré par RTL Designer Viewer	77
Annexe B — Script MATLAB : Identification expérimentale en boucle ouverte	79
Annexe C — Script MATLAB : Tracé de la réponse indicielle simulée	80
Annexe D — Fichiers de contraintes GowinEDA (.cst)	81
<i>D.1 Contraintes du projet Phase 4 (FPGA seul — test UART)</i>	81
<i>D.2 Contraintes du projet Phase 5 (SOC — FPGA + MCU + OLED)</i>	82
Annexe E — Code C : Firmware MCU Cortex-M3 (main_phase5_final.c)	82
Annexe F — Caractéristiques techniques des composants principaux	84
<i>F.1 FPGA — Tang Nano 4K (Gowin GWINSR-4C)</i>	84
<i>F.2 Moteur avec réducteur — JGA25-370</i>	85

<i>F.3 Pont en H — L298N</i>	85
<i>F.4 Afficheur OLED — SSD1306</i>	86
Annexe G — Résultats de synthèse GowinEDA	86
<i>G.1 Synthèse Phase 4 — Module PID + UART (FPGA seul)</i>	86
<i>G.2 Synthèse Phase 5 — Configuration SOC complète (FPGA + MCU)</i>	87

INTRODUCTION GÉNÉRALE

La régulation automatique constitue l'un des piliers fondamentaux de l'ingénierie moderne. Elle permet de maintenir avec précision et robustesse les grandeurs physiques caractéristiques d'un système — vitesse, position, température, pression — à des valeurs souhaitées, en dépit des perturbations extérieures et des incertitudes inhérentes à tout modèle mathématique [1]. Dans le domaine des systèmes embarqués, où les ressources de calcul sont limitées et où le déterminisme temporel est une exigence critique, la conception de boucles de régulation performantes soulève des défis spécifiques qui nécessitent une approche rigoureuse alliant automatique théorique et architecture matérielle dédiée.

Parmi les stratégies de commande disponibles, le régulateur PID (Proportionnel-Intégral-Dérivé) occupe une position dominante. Des enquêtes industrielles récurrentes indiquent de manière convergente que plus de 90 % des boucles de régulation en service dans l'industrie manufacturière et de procédés font appel à cette structure ou à l'une de ses variantes simplifiées [2]. Cette prédominance s'explique par la combinaison de trois qualités rarement réunies : une structure mathématique simple à comprendre et à mettre en œuvre, une robustesse avérée aux incertitudes de modélisation, et une capacité d'adaptation à une très vaste gamme de procédés physiques.

Ce mémoire s'inscrit dans le cadre du Master 2 Électronique des Systèmes Embarqués et porte sur la conception, l'implémentation et la validation expérimentale d'un système de régulation de vitesse par correcteur PI numérique implanté sur FPGA (Field-Programmable Gate Array). Le procédé à commander est un motoréducteur à courant continu JGA25-370 à aimants permanents, équipé d'un encodeur incrémental en quadrature pour la mesure de vitesse. La plateforme matérielle retenue est la carte Tang Nano 4K, intégrant le circuit GW1NSR-4C du fabricant Gowin Semiconductor, qui associe une matrice logique reconfigurable à un cœur processeur ARM Cortex-M3 câblé [3]. Cette architecture hybride constitue un cas d'étude particulièrement riche, car elle confronte le concepteur simultanément aux contraintes de la synthèse VHDL, au développement de firmware bare-metal, et à l'adaptation des interfaces électriques entre domaines de tension hétérogènes (1,8 V, 3,3 V et 5 V).

La démarche adoptée s'articule en quatre phases complémentaires et progressives. Une première phase théorique, développée au Chapitre I, établit les fondements de l'asservissement en boucle fermée, présente la structure mathématique du régulateur PID continu et détaille les méthodes de discrétisation pour l'implémentation numérique en arithmétique entière. Une

deuxième phase, traitée au Chapitre II, est consacrée à la présentation exhaustive de l'environnement matériel et logiciel du projet. Une troisième phase, au Chapitre III, réalise l'identification expérimentale du modèle du procédé à partir de mesures réelles sur banc d'essai, puis procède au réglage du correcteur PI et à sa validation par simulation sous MATLAB. Enfin, une quatrième phase, au Chapitre IV, décrit l'implémentation VHDL sur FPGA, le firmware du microcontrôleur Cortex-M3 et les résultats de validation expérimentale en boucle fermée.

Ce document est structuré en quatre chapitres. Le Chapitre I développe les concepts fondamentaux de l'asservissement, la structure du régulateur PID, ses méthodes de réglage et sa discrétisation pour l'implémentation numérique. Le Chapitre II présente l'intégralité des ressources matérielles et logicielles mobilisées. Le Chapitre III traite de la modélisation du motoréducteur, de l'identification expérimentale et du réglage du correcteur PI. Le Chapitre IV détaille l'architecture VHDL implantée, le firmware et la validation expérimentale du système complet.

CHAPITRE I

Asservissement, Régulateur PID et Implémentation Numérique

Introduction

La régulation automatique est la discipline de l'automatique qui vise à maintenir une grandeur physique mesurable — dite grandeur réglée — à une valeur souhaitée, en dépit des perturbations susceptibles de l'en écarter [1]. Ses applications couvrent des domaines aussi variés que la régulation de température dans les procédés industriels, le contrôle de vitesse et de position dans les systèmes électromécaniques, la régulation de débit et de pression dans les installations hydrauliques, ou encore la stabilisation d'attitude dans les systèmes aérospatiaux.

Parmi les stratégies de commande disponibles, le régulateur PID (Proportionnel-Intégral-Dérivé) occupe une place prépondérante. Sa structure tripartite, sa robustesse aux incertitudes de modèle et son efficacité prouvée sur une large gamme de procédés en font le choix de référence pour l'asservissement des systèmes linéaires monovariables [2]. Ce premier chapitre développe, de façon progressive et rigoureuse, les concepts fondamentaux nécessaires à sa mise en œuvre dans un contexte numérique embarqué : principes de l'asservissement en boucle fermée, structure et réglage du PID, discrétisation de la loi de commande, et comparaison des technologies d'implémentation disponibles.

I.1 L'Asservissement en Boucle Fermée

I.1.1 Définition et principe

Un système asservi est un système automatique dans lequel la grandeur réglée est continuellement mesurée et comparée à une valeur de référence, appelée consigne. Tout écart entre la mesure et la consigne déclenche automatiquement une action corrective tendant à le réduire. Cette architecture, dite en boucle fermée ou à contre-réaction (feedback [4]), constitue le principe fondateur de toute régulation automatique.

Elle s'oppose à la commande en boucle ouverte, dans laquelle l'action de commande est calculée sans tenir compte de l'état réel du système. Si la boucle ouverte est simple à mettre en œuvre, elle est fondamentalement incapable de compenser les perturbations extérieures ou les écarts entre le modèle mathématique et la réalité physique. La boucle fermée, au contraire,

s'autocorrige naturellement grâce au mécanisme de rétroaction, propriété qui fonde sa supériorité pratique [5].

I.1.2 Schéma fonctionnel d'un système asservi

Le schéma fonctionnel d'une boucle de régulation en contre-réaction fait intervenir les éléments suivants (Figure I.1) :

- La consigne $w(s)$: valeur souhaitée pour la grandeur réglée, fournie par l'opérateur ou par un système superviseur.
- Le comparateur : calcule le signal d'erreur $e(s) = w(s) - y(s)$, différence algébrique entre la consigne et la mesure de sortie.
- Le correcteur (régulateur) $G_c(s)$: traite le signal d'erreur $e(s)$ et élabore la commande $u(s)$ selon la loi de contrôle retenue.
- Le procédé $G_a(s)$: système physique à commander, qui transforme la commande $u(s)$ en grandeur de sortie $y(s)$.
- Le capteur et la chaîne de retour : mesurent la sortie $y(s)$ et la restituent sous une forme comparable à la consigne, fermant la boucle de régulation.

Dans le cas d'un système numérique, les signaux continus sont remplacés par leurs équivalents échantillonnés, définis aux instants discrets $t = k \cdot T_e$, où T_e est la période d'échantillonnage et k un entier naturel. Le correcteur numérique traite ces séquences et produit une commande mise à jour à chaque période T_e [4].

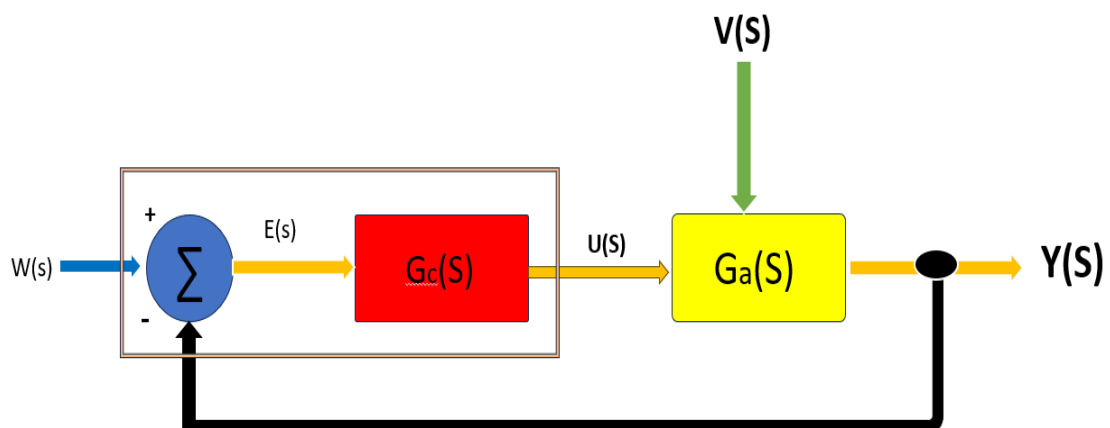


Figure I.1 — Schéma fonctionnel d'un système asservi en boucle fermée

1.1.3 Critères de performance d'un asservissement

La qualité d'un système asservi est classiquement évaluée à travers quatre critères fondamentaux [5]. Ces critères sont partiellement antagonistes, et leur équilibre constitue le problème central du réglage d'un régulateur.

a) Stabilité

La stabilité est la condition nécessaire absolue à l'utilisation d'un système asservi. Au sens BIBO (*Bounded-Input Bounded-Output*) [6], un système est stable si toute entrée bornée produit une sortie bornée. Pour un système linéaire continu invariant, cela revient à exiger que tous les pôles de la fonction de transfert en boucle fermée aient des parties réelles strictement négatives. Pour un système discret, les pôles doivent se trouver à l'intérieur du cercle unité dans le plan en Z .

Les marges de stabilité quantifient la distance au seuil d'instabilité. La marge de gain G_m (en dB) indique de combien le gain de la boucle ouverte peut augmenter avant que le système ne devienne instable. La marge de phase ϕ_m (en degrés) indique le déphasage supplémentaire tolérable à la fréquence de coupure. Des valeurs typiques $G_m > 6$ dB et $30^\circ < \phi_m < 60^\circ$ sont visées pour un bon compromis stabilité-rapidité.

b) Précision

La précision se mesure par l'erreur statique ε_s , définie comme l'écart résiduel entre la consigne et la sortie en régime permanent. Elle dépend du type du système en boucle fermée : un système de type 0 présente une erreur statique non nulle face à un échelon de consigne ; un système de type 1 — intégrant un intégrateur dans la chaîne directe — annule cette erreur pour une entrée en échelon, mais présente une erreur résiduelle face à une rampe.

c) Rapidité

La rapidité caractérise la vitesse de réaction du système aux variations de consigne ou à l'apparition d'une perturbation. Elle s'exprime par le temps de montée t_r (durée pour que la sortie passe de 10 % à 90 % de sa valeur finale) et par le temps de réponse à ± 5 %, noté t_{r5} . Un temps de réponse court est souhaitable, mais une rapidité excessive entraîne généralement un dépassement accru et une dégradation des marges de stabilité.

d) Robustesse

La robustesse est la propriété d'un système asservi à maintenir ses performances — stabilité, précision et rapidité — face aux incertitudes sur le modèle du procédé et aux variations

de ses paramètres au cours du temps [7]. C'est une exigence incontournable en pratique, où le modèle mathématique n'est jamais parfaitement exact et où les conditions opératoires évoluent.

I.2 Le Régulateur PID

I.2.1 Structure générale et équation de commande

Le régulateur PID élabore le signal de commande $u(t)$ à partir de l'erreur $e(t)$ en combinant trois actions élémentaires : une action proportionnelle à l'erreur instantanée, une action intégrale cumulant l'erreur passée, et une action dérivée anticipant la variation de l'erreur [2]. La loi de commande en temps continu — dite forme parallèle standard — s'écrit :

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) d\tau + K_d \cdot de(t)/dt \quad (I.1)$$

où K_p est le gain proportionnel, $K_i = K_p/T_i$ le gain intégral (T_i : temps intégral), et $K_d = K_p \cdot T_d$ le gain dérivé (T_d : temps dérivé). La fonction de transfert correspondante dans le domaine de Laplace s'écrit :

$$C(s) = K_p + K_i/s + K_d \cdot s = K_p \cdot (1 + 1/(T_i \cdot s) + T_d \cdot s) \quad (I.2)$$

Le pôle à l'origine ($s = 0$) introduit par le terme K_i/s élève d'une unité le type du système en boucle fermée, garantissant l'annulation de l'erreur statique vis-à-vis d'une consigne en échelon — propriété fondamentale de l'action intégrale.

I.2.2 L'action Proportionnelle

Le terme proportionnel génère une commande directement proportionnelle à l'amplitude de l'erreur courante :

$$u_P(t) = K_p \cdot e(t) \quad (I.3)$$

Son rôle principal est d'accélérer la réaction du système : plus K_p est élevé, plus la correction est rapide face à un écart de consigne. Toutefois, l'action P seule ne peut pas annuler l'erreur statique en régime permanent, et un gain trop élevé peut engendrer des oscillations et compromettre la stabilité.

I.2.3 L'action Intégrale

Le terme intégral cumule l'erreur au fil du temps :

$$u_I(t) = K_i \cdot \int_0^t e(\tau) d\tau \quad (I.4)$$

Sa vertu fondamentale est l'annulation de l'erreur statique : aussi longtemps qu'un écart persiste entre la consigne et la mesure, l'intégrateur fait croître la commande jusqu'à résorber cet écart. Son inconvénient principal est le phénomène d'emballement intégral

(*integral windup*) [8]: lorsque l'actionneur sature, l'intégrateur accumule inutilement de l'énergie, provoquant un dépassement important lors du retour en régime linéaire. Ce phénomène est systématiquement traité par un mécanisme d'anti-emballement dans toute implémentation sérieuse.

I.2.4 L'action Dérivée

Le terme dérivé agit sur la vitesse de variation de l'erreur, lui conférant un caractère anticipatif :

$$uD(t) = Kd \cdot de(t)/dt \quad (I.5)$$

En réagissant à la pente de l'erreur, l'action dérivée amortit les oscillations et réduit le dépassement, améliorant la stabilité transitoire. Son inconvénient majeur est la forte sensibilité au bruit de mesure, la dérivation amplifiant les composantes hautes fréquences [9]. En pratique, elle est toujours filtrée par un filtre passe-bas de coefficient N ($5 \leq N \leq 20$ typiquement) :

$$uD_filtré(s) = Kd \cdot N \cdot s / (s + N) \quad (I.6)$$

Une pratique courante consiste à appliquer l'action dérivée sur la mesure de sortie plutôt que sur l'erreur (dérivée de la mesure), évitant ainsi les impulsions de commande lors des changements brusques de consigne.

I.2.5 Effets combinés — Synthèse

Le Tableau I.1 résume qualitativement l'influence de chaque action sur les quatre critères de performance de la boucle de régulation [2].

Tableau I.1 — Influence qualitative de chaque action PID sur les critères de performance

Action	Temps de montée	Dépassement	Erreur statique	Stabilité
P	↓ diminue	↑ augmente	↓ non nulle	↓ si K_p excessif
I	↑ augmente	↑↑ fort	→ 0 (annulée)	Risque windup
D	↓↓ améliore	↓ réduit	Aucun effet	↑ améliore

↑ : augmentation | ↓ : diminution | ↑↑ : forte augmentation

1.2.6 Structures d'association des actions

Les trois actions peuvent être combinées selon deux topologies principales [9]. Dans la structure parallèle (forme standard), chaque terme est calculé indépendamment à partir du même signal d'erreur, puis les trois contributions sont sommées. Les paramètres K_p , K_i et K_d sont découplés, ce qui facilite le réglage indépendant de chaque action. C'est la structure retenue dans ce travail, car elle se prête naturellement à une réalisation numérique en arithmétique entière.

Dans la structure série (ou interactive), les actions sont placées en cascade, couplant les paramètres entre eux et rendant le réglage moins intuitif. Elle n'est maintenue que pour des raisons de compatibilité avec d'anciens équipements.

1.2.7 Limites du régulateur PID

Bien qu'universel, le PID n'est pas adapté à toutes les situations. Il montre ses limites face à des systèmes présentant un retard pur important par rapport à la constante de temps dominante, des dynamiques d'ordre élevé à modes oscillants multiples, des non-linéarités sévères ou des paramètres fortement variables dans le temps [7]. Dans ces cas, des stratégies avancées — commande prédictive (MPC), commande par modes glissants, logique floue, commande adaptative — doivent être envisagées.

1.3 Méthodes de Réglage du Régulateur PID

La synthèse d'un régulateur PID consiste à déterminer les trois paramètres K_p , K_i et K_d tels que la boucle fermée satisfasse les critères de performance fixés. Trois familles de méthodes sont classiquement utilisées [10].

1.3.1 Méthode de Ziegler-Nichols

Proposée en 1942 par Ziegler et Nichols [11], cette méthode empirique reste l'une des plus utilisées pour obtenir rapidement un réglage initial sans modèle mathématique du procédé. La première variante, par réponse indicielle en boucle ouverte, extrait de la courbe de réponse la pente maximale a et le retard apparent L , puis calcule les gains par des formules tabulées. La seconde variante, par oscillations limites, détermine le gain ultime K_u et la période d'oscillation T_u pour en déduire les paramètres PID. Ces méthodes fournissent un réglage agressif (dépassement typique de 20 à 30 %) servant de point de départ pour une optimisation ultérieure.

1.3.2 Méthode de placement de pôles

Cette méthode analytique consiste à choisir explicitement la position souhaitée des pôles dominants de la boucle fermée dans le plan de Laplace ou dans le plan en Z , de façon à satisfaire directement les spécifications de temps de réponse et d'amortissement [6]. Elle nécessite la connaissance du modèle mathématique du procédé et conduit à un système d'équations dont la résolution donne K_p , K_i et K_d .

1.3.3 Optimisation par simulation

Lorsqu'un modèle mathématique du procédé est disponible, l'optimisation itérative par simulation est la méthode la plus fiable et la plus flexible [12]. Les paramètres sont ajustés dans un environnement de simulation (MATLAB/Simulink ou équivalent) en observant directement les réponses temporelles de la boucle fermée, jusqu'à obtenir le compromis désiré. L'outil MATLAB PID Tuner, utilisé au Chapitre III, s'inscrit dans cette famille : il calcule les gains par optimisation numérique directe à partir du modèle identifié $G(s)$, sans hypothèse restrictive sur la valeur du retard pur.

1.4 Technologies d'Implémentation Embarquée

Le régulateur PID numérique peut être implanté selon plusieurs technologies, dont les caractéristiques influent directement sur les performances de la boucle de régulation [13]. Le choix est dicté par les exigences de l'application : fréquence de la boucle de régulation, déterminisme temporel requis, ressources disponibles et contraintes de coût.

1.4.1 Implémentation analogique

Le PID analogique, réalisé à l'aide d'amplificateurs opérationnels et de réseaux R-C, traite l'erreur en temps continu avec une latence nulle [5]. Ses inconvénients sont la sensibilité à la dérive thermique et au vieillissement des composants, la difficulté d'ajustement précis des paramètres et l'absence totale de flexibilité une fois le circuit réalisé. Il constitue néanmoins le point de référence historique.

1.4.2 Implémentation sur automate programmable (API)

Les automates programmables industriels disposent de blocs fonctionnels PID préconfigurés, paramétrables via logiciel [13]. Cette approche est très répandue en automatisation industrielle. Son principal inconvénient est le temps de cycle de l'automate, de l'ordre de la dizaine de millisecondes, qui limite la fréquence d'échantillonnage et rend cette technologie inadaptée à la régulation de systèmes à dynamique rapide.

1.4.3 Implémentation sur microcontrôleur ou DSP

L'implémentation logicielle sur microcontrôleur ou processeur de signal numérique (DSP) consiste à exécuter séquentiellement l'algorithme de récurrence, cadencé par une interruption de Timer à la période T_e [14]. Cette approche est simple à développer et modifiable à tout moment. Sa principale limitation est le caractère séquentiel du traitement : toutes les opérations arithmétiques s'enchaînent, ce qui introduit une latence non nulle et un déterminisme relatif, lié aux interruptions concurrentes.

1.4.4 Implémentation sur FPGA

L'implémentation sur FPGA (Field-Programmable Gate Array) consiste à décrire l'algorithme PID sous forme d'architecture matérielle câblée, à l'aide d'un langage de description matérielle tel que le VHDL ou le Verilog [15]. Chaque opération arithmétique est assignée à un bloc matériel dédié opérant en parallèle, synchronisé par l'horloge du circuit. L'exécution est entièrement parallèle et strictement déterministe : le temps de traitement d'un échantillon est constant et égal à quelques cycles d'horloge, quelles que soient les autres tâches en cours.

Les avantages distinctifs du FPGA pour l'implémentation du PID sont : un déterminisme temporel absolu (aucune interruption, aucune préemption), une latence minimale (inférieure à 1 μ s pour les FPGA modernes), des fréquences d'échantillonnage potentiellement très élevées (de l'ordre du MHz), et la reconfigurabilité des paramètres via des registres accessibles sans resynthèse du circuit [16]. Ces propriétés justifient le choix de cette technologie dans le présent travail.

1.4.5 Comparaison des technologies

Les différentes technologies d'implémentation sont représentées dans le tableau ci-dessous.

Tableau I.2 — Comparaison des technologies d'implémentation du régulateur PID

Critère	Analogique	API / PLC	Microcontrôleur	FPGA
Nature du traitement	Continu	Séquentiel discret	Séquentiel discret	Parallèle discret
Déterminisme	Élevé	Cyclique	Relatif	Maximum (câblé)
Latence de calcul	Nulle	ms (cycle)	μ s – ms	< 1 μ s
Reconfigurabilité	Nulle	Moyenne	Élevée (code)	Via registres
Fréq. d'échantillonnage	—	< 1 kHz	$\leq$ 100 kHz	> 1 MHz

I.5 Discrétisation du Régulateur PID

I.5.1 Nécessité du passage au domaine discret

Tout calculateur numérique traite des signaux échantillonnés, définis aux instants discrets $k \cdot T_e$. Il est donc indispensable de transposer la loi de commande continue (I.1) en une équation de récurrence numérique. Cette opération, appelée discrétisation, influe directement sur la fidélité de l'approximation et sur la stabilité de la boucle numérique résultante [4].

Le théorème de Shannon-Nyquist [17] impose une fréquence d'échantillonnage $f_e = 1/T_e$ au moins deux fois supérieure à la plus haute fréquence significative du signal. En pratique, pour les asservissements, une règle empirique couramment admise exige :

$$f_e \geq 10 \cdot \omega_{BF} / (2\pi) \quad \text{soit} \quad T_e \leq \tau_{\text{procédé}} / 10 \quad (I.7)$$

où ω_{BF} est la bande passante de la boucle fermée et $\tau_{\text{procédé}}$ la constante de temps dominante du procédé. Dans ce projet, $T_e = 10 \text{ ms}$ a été retenu, ce qui représente $T/2$ par rapport à la constante de temps identifiée $T = 19,15 \text{ ms}$, rapport classiquement acceptable en commande numérique.

I.5.2 Méthodes de discrétisation

Trois méthodes sont classiquement utilisées pour approcher les opérations d'intégration et de dérivation en temps discret [4]. Le Tableau I.3 les compare sur leurs propriétés essentielles.

Tableau I.3 — Comparaison des méthodes de discrétisation du régulateur PID

Méthode	Approximation de s	Propriétés	Usage recommandé
Euler avant	$s \approx (z-1)/T_e$	Simple ; peut être instable si T_e grand	Systèmes lents, T_e très petit
Euler arrière	$s \approx (z-1)/(z \cdot T_e)$	Stable ; introduit un retard supplémentaire	Bonne robustesse en stabilité
Tustin (bilinéaire)	$s \approx (2/T_e) \cdot (z-1)/(z+1)$	Meilleure fidélité fréquentielle ; stable si continu stable	Référence industrielle

T_e : période d'échantillonnage z : variable de la transformée en Z

La méthode d'Euler avant est retenue dans ce travail pour sa simplicité d'implémentation directe en arithmétique entière [14]. Elle donne les approximations suivantes :

$$\int_0^{kTe} e(\tau) d\tau \approx Te \cdot \sum_{i=0}^k e(i) \quad (I.8)$$

$$de(t)/dt \big|_{t=kTe} \approx [e(k) - e(k-1)] / Te \quad (I.9)$$

I.5.3 Forme position et forme récurrente

En substituant les approximations (I.8) et (I.9) dans l'équation (I.1), on obtient la forme position du PID discret :

$$u(k) = Kp \cdot e(k) + Ki \cdot Te \cdot \sum_{i=0}^k e(i) + (Kd/Te) \cdot [e(k) - e(k-1)] \quad (I.10)$$

La somme infinie du terme intégral étant peu pratique, on calcule l'incrément $\Delta u(k) = u(k) - u(k-1)$ pour obtenir la forme récurrente (ou incrémentale) :

$$\Delta u(k) = Kp \cdot \Delta e(k) + Ki \cdot Te \cdot e(k) + (Kd/Te) \cdot [e(k) - 2 \cdot e(k-1) + e(k-2)] \quad (I.11)$$

avec $\Delta e(k) = e(k) - e(k-1)$. La commande est ensuite reconstruite par accumulation avec saturation :

$$u(k) = \text{sat}[u(k-1) + \Delta u(k)] = \max(\text{umin}, \min(\text{umax}, u(k-1) + \Delta u(k))) \quad (I.12)$$

La saturation dans l'équation (I.12) constitue le mécanisme d'anti-emballement [8] : en bornant la commande à la plage physiquement réalisable $[\text{umin}, \text{umax}]$ à chaque pas de calcul, elle empêche l'accumulateur intégral de dériver hors de la zone linéaire, prévenant tout emballement. Dans le code VHDL de boucle fermée (module *phase4_top_pid_bf*), ce mécanisme est implanté par gel du registre *terme_i_reg* lorsque *duty_cycle* atteint l'une de ses bornes $[\text{DUTY_MIN_C} ; \text{DUTY_MAX_C}] = [43 ; 252]$.

La forme récurrente présente plusieurs avantages majeurs pour l'implémentation numérique [14] : elle ne nécessite que les trois dernières valeurs d'erreur $e(k)$, $e(k-1)$ et $e(k-2)$, stockées dans trois registres ; elle n'implique aucune valeur initiale particulière ; et elle intègre naturellement la saturation anti-emballement. Elle ne fait appel qu'à des additions, soustractions et multiplications, directement réalisables en virgule fixe.

I.5.4 Algorithme complet du PID numérique

L'algorithme exécuté à chaque instant d'échantillonnage $k \cdot Te$ (Figure I.2) se résume aux étapes suivantes :

- Lecture de la mesure $s(k)$ depuis le capteur (registre *speed_reg* dans le module VHDL).
- Calcul de l'erreur : $e(k) = c(k) - s(k)$.
- Calcul de l'incrément de commande $\Delta u(k)$ par l'équation (I.11).
- Mise à jour avec saturation anti-emballement : $u(k) = \text{sat}[u(k-1) + \Delta u(k)]$ par l'équation (I.12).

- Envoi de la commande $u(k)$ à l'actionneur (registre `duty_cycle` vers le générateur PWM).
- Décalage des registres mémoire : $e(k-2) \leftarrow e(k-1)$, $e(k-1) \leftarrow e(k)$, $u(k-1) \leftarrow u(k)$.
- Attente de la prochaine impulsion `tick_pulse` (période $T_e = 10 \text{ ms}$ à 27 MHz).

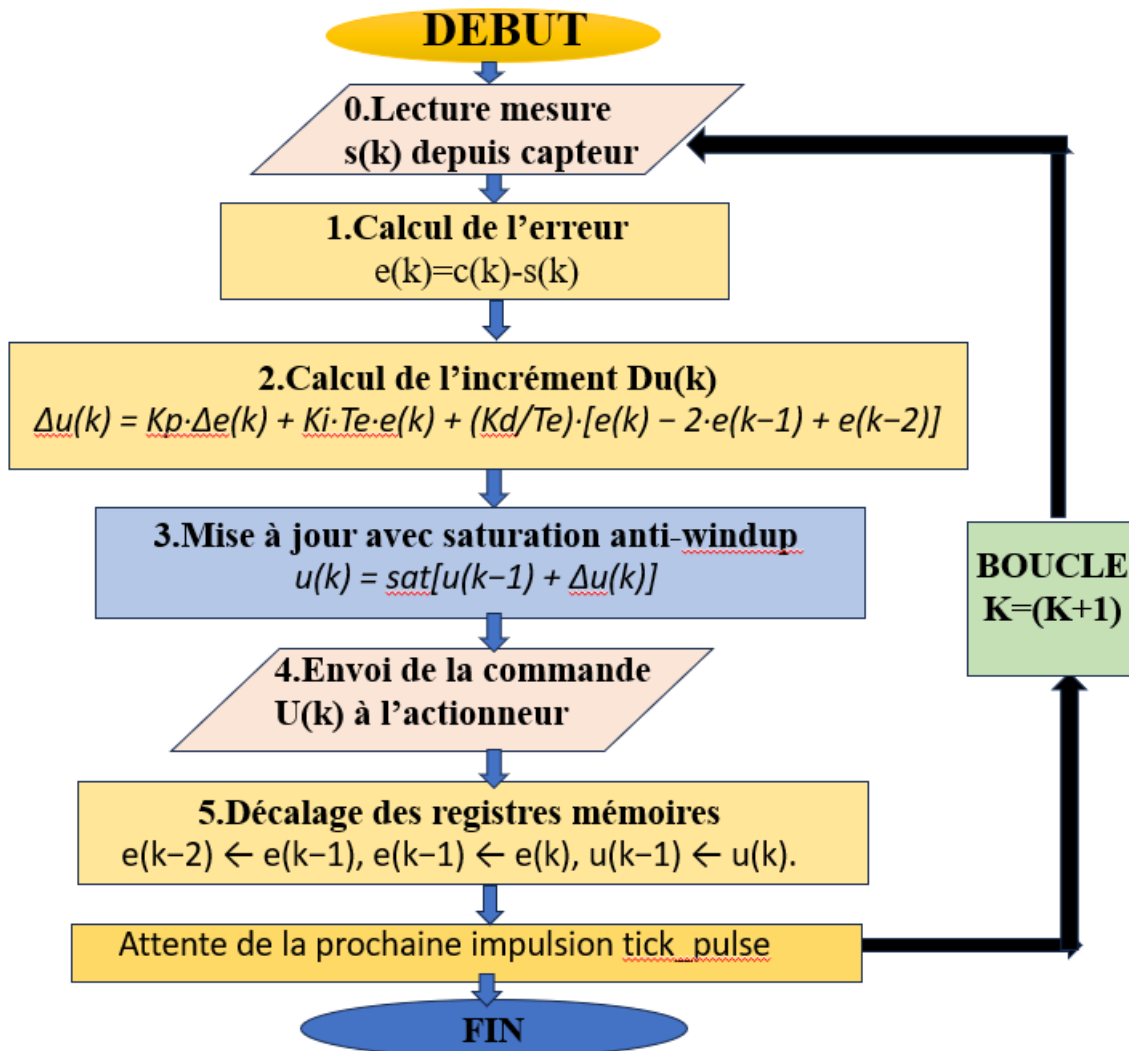


Figure I.2 — Organigramme de l'algorithme du PID numérique discret

I.6 Représentation en Virgule Fixe

I.6.1 Principe du format virgule fixe

Les calculateurs embarqués sans unité flottante matérielle traitent les données en arithmétique entière ou en virgule fixe [18]. La représentation en virgule fixe Q_n consiste à exprimer un nombre réel x sous la forme d'un entier signé, en affectant implicitement une position fixe à la virgule : $x_{\text{Réel}} = x_{\text{Entier}} \times 2^{-n}$. Le format **Q15** (1 bit de signe + 15 bits fractionnaires sur 16 bits) permet de représenter des nombres dans $[-1, +1[$ avec une résolution de $2^{-15} \approx 3,05 \times 10^{-5}$.

Deux problèmes doivent être anticipés lors de l'implémentation en virgule fixe :

- Le débordement (overflow) : le résultat d'une opération dépasse la plage représentable. Sans saturation, un retournement de signe catastrophique se produit. Dans le module VHDL, l'accumulateur `terme_i_reg` est déclaré sur 32 bits signés et borné dans $[-4000 ; +4000]$ pour prévenir ce phénomène.
- L'erreur de quantification : l'arrondi lors des multiplications introduit une erreur systématique d'amplitude au plus $2^{-n}/2$ (un demi-LSB). Elle est généralement négligeable pour $n \geq 12$.

I.6.2 Conversion des paramètres PID et facteur d'échelle SCALE

Les paramètres **Kp**, **Ki** et **Kd**, issus du réglage continu, sont convertis en entiers pour être stockés dans les registres du calculateur [18]. Dans ce projet, un facteur d'échelle entier $SCALE = 16 = 2^4$ est utilisé à la place du format **Q15** classique. Ce choix délibéré permet de remplacer la division par $SCALE$ par un simple décalage arithmétique à droite de 4 positions (*shift_right(x, 4)*), opération sans coût en ressources FPGA et garantissant une latence nulle :

$$KP_VHDL = round(Kp \times SCALE) \quad \text{avec } SCALE = 16 = 2^4 \quad (I.13)$$

$$KI_VHDL = round(Ki_discret \times SCALE) \quad (I.14)$$

$$KD_VHDL = round(Kd_discret \times SCALE) \quad (= 0 \text{ dans ce projet}) \quad (I.15)$$

Les multiplications entre mots signés 16 bits produisent un résultat sur **30 bits** significatifs, redimensionne en 32 bits par `resize(erreur_KP, 32)` pour éviter l'erreur EX4923. La démonstration formelle de la correspondance entre les paramètres MATLAB continus et les constantes entières VHDL est développée au Chapitre III, section III.8.

Conclusion

Ce chapitre a établi les fondements théoriques nécessaires à la mise en œuvre du régulateur PID dans le contexte numérique embarqué de ce projet. En partant des principes de base de la boucle de régulation en contre-réaction et des critères de performance qui la caractérisent [1,5], la structure du PID a été présentée avec le rôle de chacune de ses trois actions et les effets de leur combinaison sur le comportement dynamique de la boucle.

L'étape centrale est la discrétisation : la transposition de la loi de commande continue (I.1) en algorithme numérique, aboutissant à la forme récurrente du PID discret (I.11) et au mécanisme d'anti-emballement par saturation (I.12). Cet algorithme, exprimé en

arithmétique entière avec facteur d'échelle $SCALE = 16$, constitue la base directe de l'implémentation VHDL développée au Chapitre IV [14,15].

Le tableau comparatif des technologies d'implémentation (Tableau I.2) a mis en évidence les avantages spécifiques du FPGA — déterminisme absolu, latence minimale, parallélisme câblé et reconfigurabilité des paramètres [16] — qui justifient le choix de cette technologie (annexe A). Le Chapitre II qui suit présente l'ensemble du matériel et des logiciels déployés pour concrétiser ces concepts sur le banc d'essai réel.

CHAPITRE II

Présentation des Matériels et Logiciels Utilisés

II.1 Introduction

La réalisation d'un système embarqué de commande en vitesse par régulateur PID sur FPGA mobilise un ensemble hétérogène de composants matériels et d'environnements logiciels, dont la maîtrise individuelle et la compatibilité mutuelle conditionnent directement la réussite du projet. Ce chapitre dresse la présentation détaillée de chaque élément de la chaîne instrumentale, en précisant ses caractéristiques techniques pertinentes pour l'application, son rôle fonctionnel dans le système, et les contraintes qu'il impose à la conception.

Les choix technologiques retenus résultent d'un double impératif : disposer d'un environnement de développement cohérent permettant d'aller de l'identification expérimentale du moteur jusqu'à la synthèse et la validation du régulateur numérique sur cible réelle, tout en restant dans les limites des ressources disponibles dans un contexte académique. La section II.8 présente un bilan synthétique de l'ensemble de la chaîne matérielle.

II.2 Plateforme FPGA : Tang Nano 4K (GW1NSR-4C)

II.2.1 Architecture du circuit intégré

La carte Tang Nano 4K est basée sur le circuit intégré GW1NSR-4C du fabricant Gowin Semiconductor. Il s'agit d'un System-on-Chip (SoC) original qui associe dans un même boîtier deux domaines de calcul distincts et complémentaires : une matrice de logique programmable (FPGA fabric) et un microcontrôleur ARM Cortex-M3 câblé (hard IP)[3].

Cette architecture hybride présente un intérêt majeur pour les applications de commande embarquée. Le domaine FPGA assure le traitement temps réel strict des boucles de régulation, de la génération PWM et du décodage de l'encodeur, fonctions nécessitant une précision temporelle à l'échelle de la dizaine de nanosecondes. Le domaine MCU prend en charge les tâches moins critiques en termes de latence, telles que la communication série et la gestion de l'affichage, libérant ainsi les ressources logiques du FPGA [3,19].

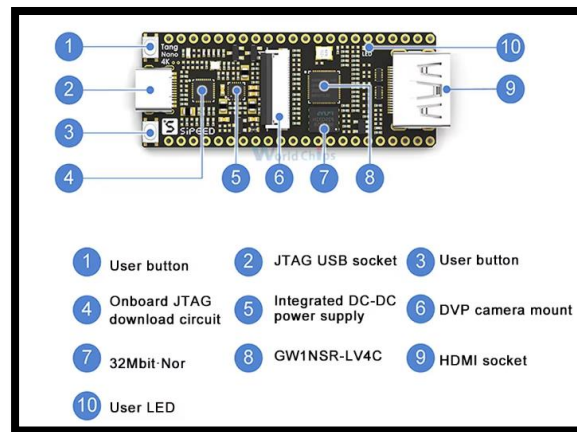


Figure II.1 — Carte de développement Tang Nano 4K

II.2.2 Ressources logiques et caractéristiques électriques

Le Tableau II.1 résume les ressources internes du GW1NSR-4C pertinentes pour ce projet.

Tableau II.1 — Ressources du circuit GW1NSR-4C exploitées dans le projet

Ressource	Quantité disponible	Utilisation dans le projet
Cellules logiques LUT4	4 608	Logique de commande, filtres, UART
Bascules (Flip-Flops)	4 608	Registres d'état, pipelines
Blocs DSP18 (multiplicateurs câblés)	4	Multiplications PID (erreur $\times$ KP, erreur $\times$ KI)
Blocs RAM (BSRAM 18 kb)	4	Mémoire programme MCU (firmware)
PLL (Gowin_PLLVR)	2	Génération 54 MHz pour Cortex-M3
Microcontrôleur ARM Cortex-M3	1 (hard IP)	Firmware affichage OLED, parsing UART
Tension d'alimentation des E/S	1,8 V (LVCMOS18)	Toutes les broches du projet
Fréquence d'horloge principale	27 MHz (oscillateur embarqué)	FPGA fabric — base de tous les compteurs

La tension de fonctionnement de 1,8 V des entrées/sorties constitue la contrainte électrique la plus structurante du projet. Elle impose l'emploi de diviseurs de tension ou de

convertisseurs de niveau logique pour toute interface avec des composants fonctionnant en logique 3,3 V ou 5 V, notamment l'encodeur et la liaison UART vers l'Arduino Uno.

II.2.3 Programmation et débogage

La configuration du FPGA est réalisée par l'environnement GowinEDA via l'interface USB intégrée à la carte (puce FTDI YP-05). La même interface physique assure deux fonctions simultanées : le transfert du bitstream de configuration FPGA et la communication UART série avec le PC hôte. Le port série apparaît sous Windows comme un port COM virtuel à 115 200 bauds, exploité pour la supervision en temps réel des trames de télémétrie. Le microcontrôleur Cortex-M3 est programmé séparément via l'environnement Gowin MCU SDK basé sur Eclipse, le binaire étant intégré dans le bitstream final comme initialisation des blocs RAM. [20]

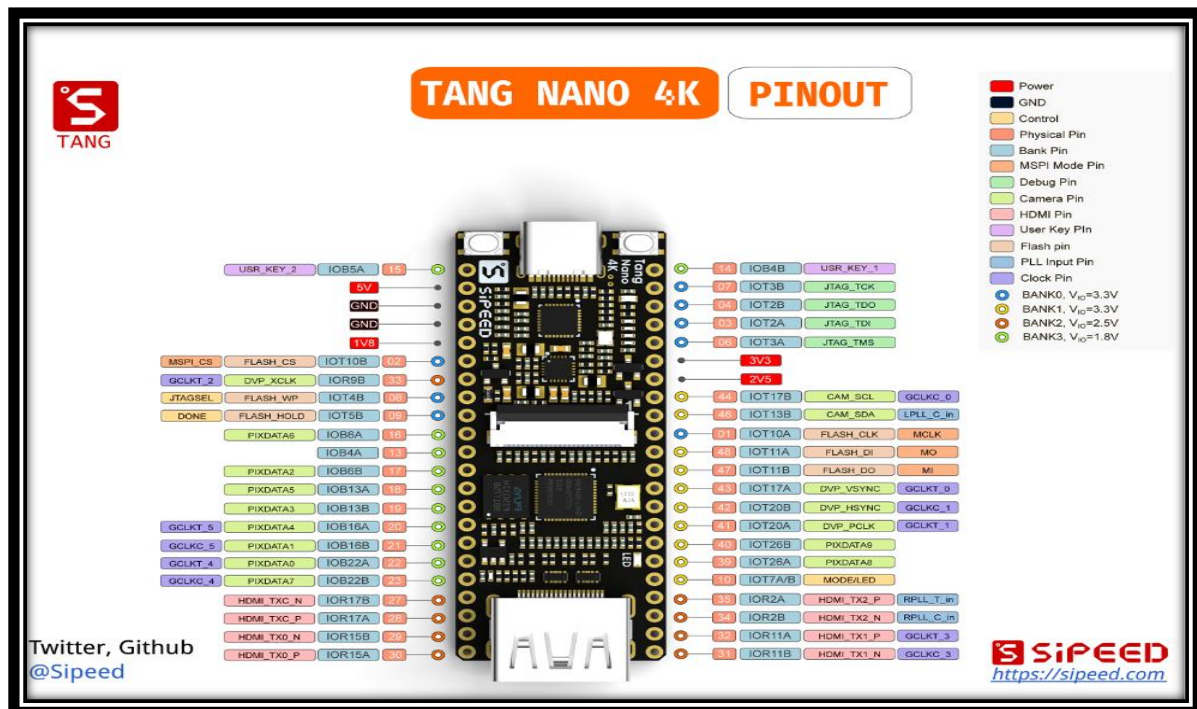


Figure II.2 — Interface USB-JTAG intégrée et brochage de la Tang Nano 4K

II.3 Actionneur : Motoréducteur JGA25-370

II.3.1 Description et caractéristiques

Le motoréducteur JGA25-370 est un composant électromécanique compact largement utilisé dans les applications de robotique et d'automatismes embarqués à faible puissance. Il intègre dans un boîtier cylindrique de 25 mm de diamètre trois sous-systèmes fonctionnels indissociables : un moteur à courant continu à aimants permanents, un réducteur planétaire

multi-étages, et un encodeur incrémental en quadrature. Le Tableau II.2 en rappelle les caractéristiques principales.

Tableau II.2 — Caractéristiques du motoréducteur JGA25-370

Paramètre	Valeur	Remarque
Tension nominale	12 V	Alimentation externe
Courant à vide	≈ 150 mA	Sans charge
Rapport de réduction	4,45 : 1	Réducteur planétaire
Résolution encodeur (arbre moteur)	11 PPR — mode x1	Comptage fronts montants voie A
Résolution encodeur (arbre sortie)	49 PPR — mode x1	$11 \times 4,45 \approx 49$
Tension alimentation encodeur	5 V	Signaux de sortie en 5 V logique
Type de signaux encodeur	Quadrature A / B	Déphasage 90°

II.3.2 Rôle de l'encodeur dans la boucle de régulation

L'encodeur incrémental constitue l'unique capteur de retour de la boucle de régulation. Les deux signaux en quadrature (voies A et B) permettent de déterminer simultanément la vitesse et le sens de rotation. Dans ce projet, le décodage est réalisé en mode x1 : seuls les fronts montants de la voie A sont comptés, ce qui donne une résolution de 11 impulsions par tour sur l'arbre moteur. La mesure de vitesse est obtenue par comptage des impulsions sur une fenêtre temporelle fixe de 10 ms, produisant une valeur entière directement exploitable par le correcteur PI numérique.

La résolution de 11 impulsions par tour constitue la principale limitation métrologique du système à basse vitesse : une variation d'une impulsion représente une erreur relative de 100 % à la vitesse minimale (1 impulsion par fenêtre). Ce fait justifie la délimitation de la plage opérationnelle à [1 ; 10] impulsions par fenêtre de 10 ms, analysée en détail au Chapitre IV.

II.3.3 Adaptation des niveaux de tension

Les signaux de l'encodeur étant délivrés en logique 5 V alors que les entrées du FPGA opèrent en LVCMOS18 (1,8 V), une adaptation de niveau est obligatoire pour éviter tout endommagement du circuit GW1NSR-4C. Cette adaptation est réalisée par un simple diviseur

résistif placé sur chacune des voies A et B avant les broches PIN 39 et PIN 40 du FPGA. Le filtre numérique interne au VHDL (compteur borné à 150 cycles) élimine les rebonds résiduels introduits par les résistances.

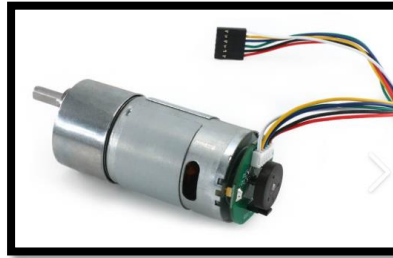


Figure II.3 — Motoréducteur JGA25-370

II.4 Interface de Puissance : Pont en H L298N

II.4.1 Principe de fonctionnement

Le L298N est un circuit intégré de puissance de type double pont en H, conçu pour piloter des charges inductives bidirectionnelles (moteurs CC, moteurs pas à pas) à partir de signaux de commande en logique TTL ou CMOS. Le principe du pont en H repose sur la commutation de quatre transistors bipolaires NPN agencés en pont de Wheatstone : selon la combinaison des états logiques sur les entrées IN1 et IN2, le courant traverse le moteur dans un sens ou dans l'autre, permettant le contrôle du sens de rotation [22].

La modulation de la vitesse est assurée par application d'un signal PWM sur l'entrée ENA. Le rapport cyclique du signal PWM détermine la valeur moyenne de la tension aux bornes du moteur et, par conséquent, la vitesse de rotation. La fréquence PWM retenue est de 10 kHz, valeur suffisamment élevée pour que la constante de temps mécanique du motoréducteur ($T = 19,15$ ms) lisse les variations instantanées de couple [23].

II.4.2 Caractéristiques électriques et contraintes

Le L298N supporte une tension d'alimentation moteur jusqu'à 46 V et un courant de sortie continu de 2 A par canal. Dans ce projet, l'alimentation moteur est fixée à 12 V. La contrainte principale du L298N est sa chute de tension interne d'environ 2 V à pleine charge, qui réduit la tension effective aux bornes du moteur de 12 V à environ 10 V à duty cycle maximal. Cette non-linéarité a été prise en compte dans la démonstration formelle du changement d'échelle présentée au Chapitre III : elle explique la réduction du gain effectif de boucle d'un facteur $\approx 0,5$ et justifie l'écart entre le gain KP théorique (27) et la valeur validée expérimentalement (13).

Tableau II.3 — Paramètres de configuration du L298N dans le projet

Signal / Paramètre	Valeur	Broche FPGA
ENA (entrée PWM)	PWM 10 kHz, duty [43;252]/255	PIN 27
IN1 (direction)	Logique 1,8 V	PIN 28
IN2 (direction, complément)	NOT(IN1)	PIN 29
Tension alimentation moteur	12 V	Alimentation externe
Chute de tension interne	≈ 2 V	Non-linéarité à modéliser
Rapport cyclique minimal (DUTY_MIN)	43 / 255	Seuil de démarrage moteur
Rapport cyclique maximal (DUTY_MAX)	252 / 255	Marge sécurité 3 unités

II.4.3 Protection contre les inversions de sens : temps mort

L'inversion du sens de rotation impose de basculer simultanément les états logiques sur IN1 et IN2. Sans précaution, la commutation peut créer un court instant où les deux entrées sont actives, provoquant un court-circuit entre l'alimentation et la masse à l'intérieur du pont. Pour prévenir ce risque, le code VHDL insère un temps mort de 100 cycles d'horloge (soit $3,7 \mu\text{s}$ à 27 MHz) pendant lequel le signal PWM est forcé à zéro avant tout changement d'état des signaux de direction. Ce mécanisme est implanté dans le processus dédié dead-time du module `phase4_top`.

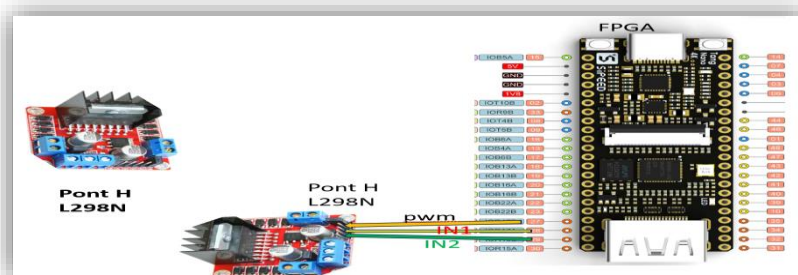


Figure II.4 — Module L298N et schéma de câblage avec le FPGA

II.5 Instruments de Mesure et de Débogage

II.5.1 Analyseur logique *PulseView*

L'analyseur logique constitue l'instrument central de validation tout au long du projet. Il remplace avantageusement l'oscilloscope pour les signaux numériques multi-canaux, en permettant la capture synchronisée de plusieurs signaux avec une résolution temporelle de 250 ns. Son logiciel *PulseView* offre une interface de décodage protocolaire intégrée (PWM, UART, I2C, SPI) et une fonction d'exportation au format CSV, exploitée par le script MATLAB d'identification en boucle ouverte.

Dans ce projet, l'analyseur logique est utilisé à trois stades distincts : lors de la phase d'identification pour capturer simultanément le signal PWM et les voies A et B de l'encodeur afin de construire le fichier `capture_bo.csv` ; lors de la validation du correcteur PI pour observer en temps réel l'évolution du registre `speed_reg` et la convergence vers la consigne ; et lors de l'intégration SoC pour vérifier la trame UART à 115 200 baud.



Figure II.5 — analyseur logique à 8 entrées

II.5.2 Multimètre

Le multimètre est utilisé pour deux vérifications électriques élémentaires mais indispensables. Premièrement, la mesure de la tension moyenne en sortie du L298N sur la borne ENA confirme la proportionnalité entre le rapport cyclique programmé et la tension effective appliquée au moteur, permettant de quantifier la chute de tension interne du pont. Deuxièmement, la vérification des niveaux de tension sur les broches d'entrée du FPGA (voies encodeur) avant branchement garantit l'absence de dépassement du seuil LVCMOS18.

II.5.3 Terminal PuTTY

Le terminal PuTTY est configuré à 115 200 bauds, 8 bits de données, sans parité, 1 bit de stop (format 8N1). Il permet la visualisation en temps réel des trames ASCII émises par le FPGA sur la broche PIN 17 via l'interface FTDI de la carte. Chaque trame au format `S+05,C05,P13,I24,E+00` est reçue à une cadence de 100 Hz, permettant de surveiller la dynamique de régulation (vitesse, consigne, erreur, gains) pendant les essais de validation. L'absence d'erreurs de caractères ou de pertes d'octets sur plusieurs minutes de fonctionnement confirme la stabilité du générateur de trames VHDL.

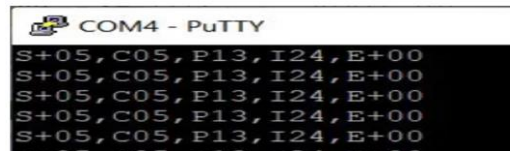


Figure II.6 — Terminal PuTTY

II.6 Affichage : Écran OLED SSD1306 et Arduino Uno

II.6.1 Écran OLED SSD1306 — Caractéristiques et contrainte de tension

L'écran OLED SSD1306 est un afficheur graphique monochromatique de résolution 128×64 pixels, piloté par le protocole I2C à l'adresse 0x3C. Sa faible consommation (< 20 mA) et sa taille compacte en font un composant standard des systèmes embarqués à faible encombrement.

Dans ce projet, le pilotage direct de l'écran par le microcontrôleur Cortex-M3 du GW1NSR-4C s'est heurté à une incompatibilité fondamentale de niveaux logiques. Le SSD1306 exige un niveau haut $V_{IH} \geq 2,64$ V sur ses lignes SDA et SCL, alors que les sorties GPIO du Cortex-M3 intégré ne délivrent que 1,8 V. Cette incompatibilité, mise en évidence lors de la phase d'intégration SoC, exclut toute connexion directe et aurait nécessité un convertisseur de niveau bidirectionnel (BSS138 ou TXS0108E), non disponible lors du projet. Cette contrainte a conduit à l'architecture de relais présentée en section II.6.3 [24].

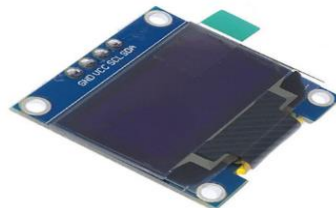


Figure II.7 — Écran OLED SSD1306 de résolution 128×64 pixels

II.6.2 Arduino Uno — Rôle de convertisseur de protocole

L'Arduino Uno, basé sur le microcontrôleur ATmega328P alimenté en 5 V, est utilisé comme convertisseur de protocole entre la liaison UART reçue du FPGA et le protocole I2C attendu par l'écran OLED. Cette solution de substitution présente l'avantage d'être immédiatement opérationnelle avec les bibliothèques disponibles dans l'environnement Arduino IDE [25]

Le firmware Arduino embarque la bibliothèque U8g2 pour le pilotage de l'écran SSD1306. À chaque réception d'une trame UART complète (délimitée par le caractère de retour chariot `\r\n`), le firmware parse les cinq champs de la trame et met à jour l'affichage OLED avec un rafraîchissement de 500 ms. Les grandeurs affichées sont : la vitesse mesurée, la consigne courante, les gains KP et KI, et l'erreur de régulation [26].

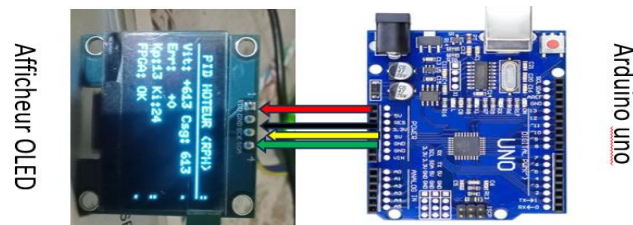


Figure II.8 — Arduino Uno utilisé comme relais UART → I2C vers l'écran OLED

II.6.3 Convertisseur de niveau logique 1,8 V → 5 V

La liaison UART entre le FPGA (1,8 V) et l'Arduino Uno (5 V) nécessite une élévation du niveau logique en émission. Un convertisseur de niveau logique unidirectionnel est inséré entre la broche PIN 17 du FPGA (UART TX, 1,8 V) et la broche RX de l'Arduino (5 V). Ce composant assure que le niveau haut du signal UART émis par le FPGA (1,8 V) est correctement interprété comme un niveau haut par l'entrée TTL de l'Arduino ($V_{IH_min} = 3\text{ V}$).

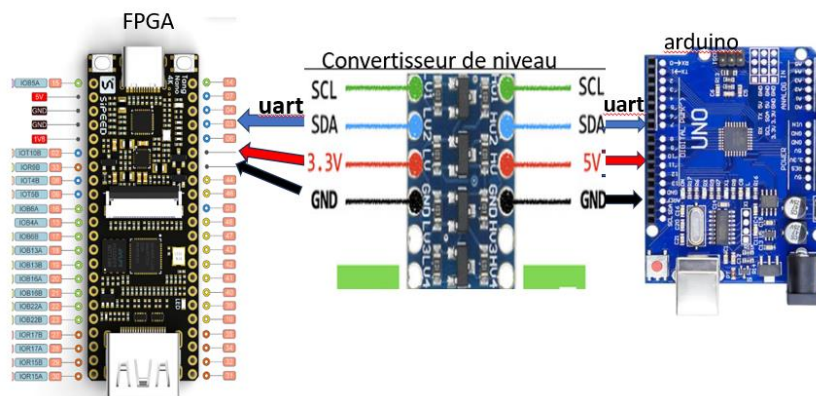


Figure II.9 — Schéma complet de la chaîne d'affichage FPGA → convertisseur → Arduino → OLED

II.7 Environnements Logiciels

II.7.1 GowinEDA

GowinEDA est l'environnement de développement intégré fourni par Gowin Semiconductor pour la conception sur ses circuits FPGA. Il recouvre l'ensemble du flot de conception numérique : édition et vérification syntaxique du code VHDL, synthèse logique (conversion du VHDL en netliste de primitives Gowin), implémentation (placement et routage

des ressources sur la matrice du GW1NSR-4C), analyse temporelle statique (rapport de slack sur les chemins critiques), et génération du fichier bitstream de configuration [20].

Parmi les contraintes spécifiques de GowinEDA qui ont influencé les choix d'architecture VHDL dans ce projet, on retiendra : l'absence de support des constructions VHDL-2008 telles que le mot-clé `wait` dans les processus synchrones ; l'obligation de respecter la règle des pilotes uniques (un seul processus autorisé à affecter un signal donné, sous peine d'erreur CV0013) ; et la nécessité de traiter explicitement les largeurs de bits dans les opérations arithmétiques signées pour éviter les avertissements EX4923 lors de la synthèse.

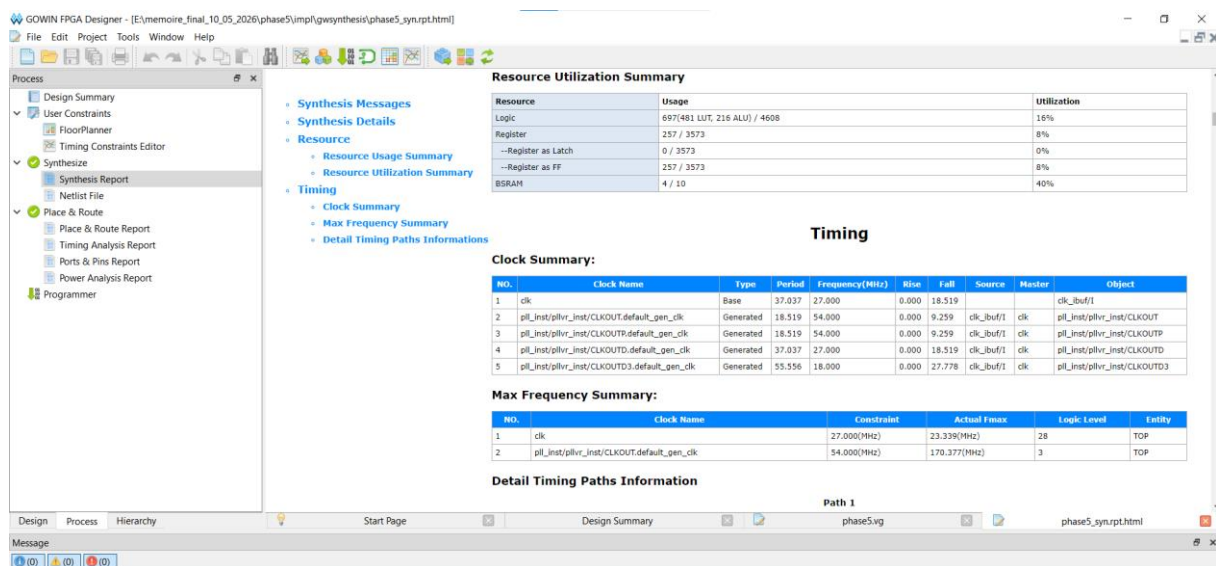


Figure II.10 — Interface GowinEDA : fenêtre de synthèse et rapport d'utilisation des ressources

II.7.2 Gowin MCU SDK (Eclipse)

Le Gowin MCU SDK est l'environnement de développement dédié au microcontrôleur Cortex-M3 intégré dans le GW1NSR-4C. Il est basé sur l'IDE Eclipse avec un plugin Gowin et la chaîne de compilation ARM GCC. Il fournit les fichiers d'en-tête de périphériques spécifiques au GW1NSR-4C (registres GPIO, UART, timers), les scripts de liaison (linker scripts) adaptés à la carte mémoire interne, et un outil de débogage JTAG [20,27].

Le firmware développé dans ce projet est un programme C bare-metal (sans système d'exploitation) qui initialise la PLL à 54 MHz, configure le récepteur UART, et implante la logique de parsing de la trame ASCII et de pilotage de l'écran OLED par bit-banging I2C sur les broches GPIO[2] et GPIO[3]. Trois corrections ont été apportées lors de l'intégration, liées à des doublons de fonctions, à une API GPIO inexistante dans le SDK, et à des stubs newlib manquants, détaillées au Chapitre IV.

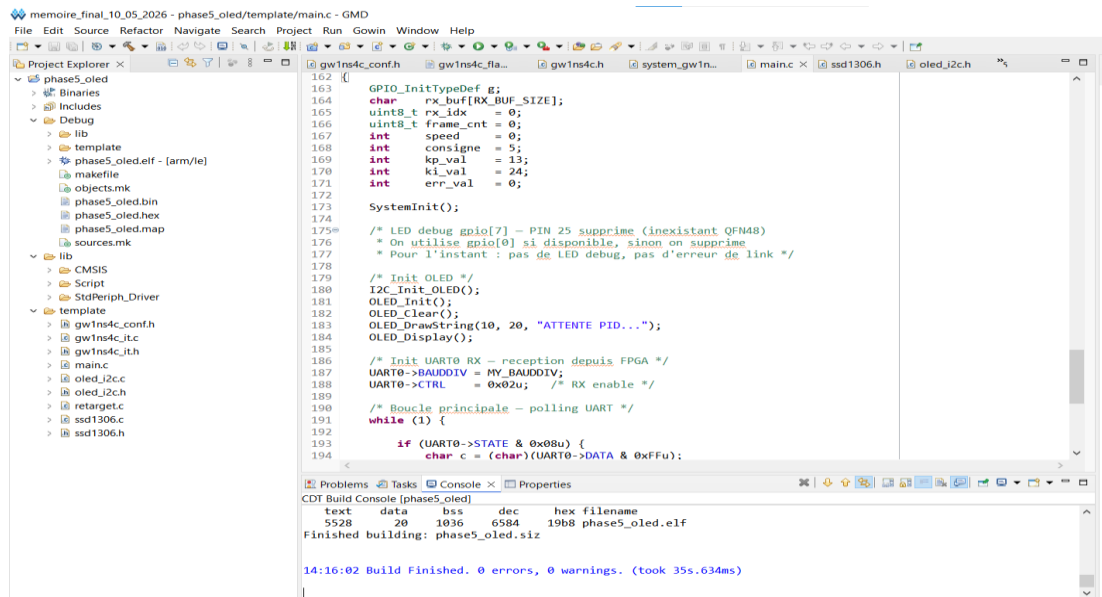


Figure II.11 — Environnement Eclipse / Gowin MCU SDK

II.7.3 MATLAB et MATLAB PID Tuner

MATLAB est utilisé à deux stades du projet avec des rôles bien distincts. En phase d'identification, le script d'analyse `identification_bo.m` traite les fichiers CSV exportés par l'analyseur logique PulseView. Il reconstruit la commande normalisée, détecte automatiquement l'instant de l'échelon, calcule la vitesse par comptage de fronts, et ajuste les paramètres du modèle du premier ordre par optimisation non linéaire (fminsearch, algorithme de Nelder-Mead). En phase de réglage, la fonction `pidTuner(G, 'PI')` lance l'interface graphique PID Tuner à partir de la fonction de transfert identifiée. Le curseur Transient Behavior est positionné à 0,80 (zone Robuste) pour privilégier l'absence de dépassement, conformément aux exigences d'une commande de moteur embarqué [28,29].

La Control System Toolbox est ensuite utilisée pour simuler la boucle fermée avec le correcteur PI discret, obtenu par discrétisation de Tustin à $T_e = 10$ ms, et pour valider les marges de stabilité avant implantation sur le FPGA [1,28].

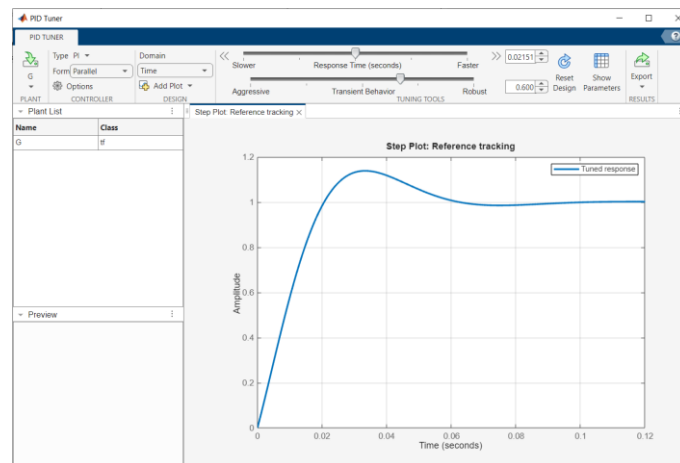


Figure II.12 — Interface MATLAB PID Tuner

II.7.4 PulseView

PulseView est le logiciel d'acquisition et d'analyse associé à l'analyseur logique. Il permet la configuration du taux d'échantillonnage (250 ns dans ce projet), la visualisation simultanée de plusieurs canaux logiques, le décodage automatique des protocoles (UART, I2C, PWM), et l'exportation des données brutes au format CSV. Ce fichier CSV constitue l'entrée du pipeline de traitement MATLAB. La conversion des horodatages en nanosecondes vers des secondes est effectuée automatiquement dans le script MATLAB, conformément aux en-têtes de colonnes détectées [30].

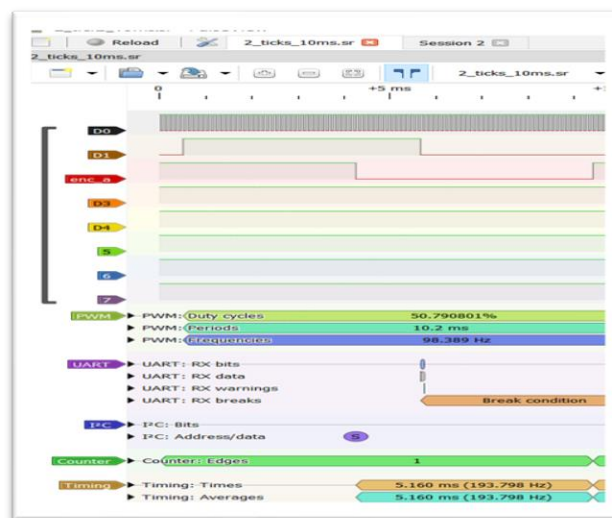


Figure II.13 — interface du logiciel PulseView

II.7.5 PuTTY

PuTTY est un émulateur de terminal série open source disponible sous Windows. Dans ce projet, il est configuré en connexion série directe sur le port COM virtuel créé par l'interface FTDI de la Tang Nano 4K, à 115 200 bauds, format 8N1. Il constitue l'outil de débogage de la

liaison UART en temps réel, permettant de visualiser les trames ASCII émises par le FPGA et de vérifier la cohérence des données de télémétrie (vitesse mesurée, consigne, gains, erreur) pendant les essais de validation en boucle fermée [31].

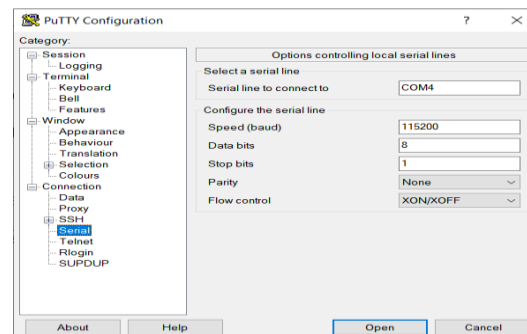


Figure II.14 — interface logiciel du terminal PuTTY

II.8 Récapitulatif de la Chaîne Matérielle Complète

Le Tableau II.4 présente le bilan complet des composants matériels impliqués dans le système, avec leur rôle fonctionnel, leur interface électrique et leur tension de fonctionnement.

Tableau II.4 — Bilan matériel de la chaîne de commande

Composant	Rôle	Interface	Tension	Connexion
Tang Nano 4K (GW1NSR-4C)	FPGA + Cortex-M3 : contrôleur central	JTAG / USB	1,8 V E/S	PC (USB)
L298N	Interface de puissance moteur	PWM + DIR	12 V moteur / 5 V logique	PIN 27/28/29
JGA25-370	Actionneur + capteur vitesse	Alimentation + encodeur	12 V / 5 V encodeur	L298N + FPGA
Diviseur de tension (R1/R2)	Adaptation 5 V → 1,8 V encodeur	Passif (résistances)	5 V → 1,8 V	PIN 39/40
Convertisseur logique UART	Adaptation 1,8 V → 5 V UART TX	Unidirectionnel	1,8 V → 5 V	PIN 17 → Uno RX
Arduino Uno (ATmega328P)	Relais UART → I2C / affichage	UART RX + I2C	5 V	UART + SSD1306
Écran OLED SSD1306	Affichage télémétrie temps réel	I2C (0x3C)	3,3 V / 5 V	Arduino I2C
Analyseur logique Saleae	Acquisition signaux / export CSV	Sondes logiques	1,8 V – 5 V	Multicanal
PC hôte (Windows)	GowinEDA, MATLAB, PuTTY	USB	—	USB-JTAG

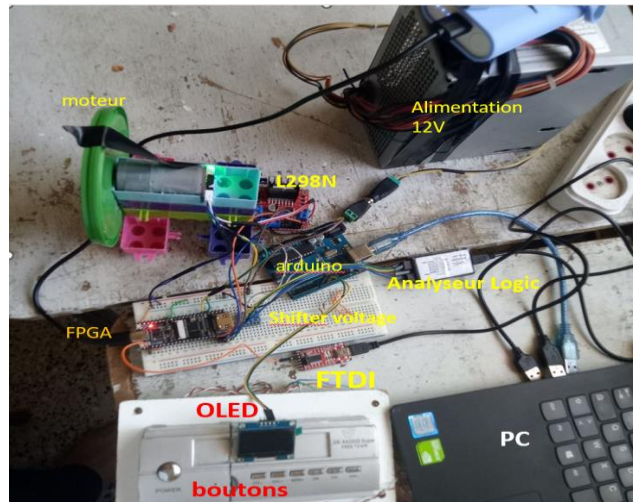


Figure II.15 — Photo du banc d'essai complet

II.9 Conclusion

Ce chapitre a présenté l'ensemble des ressources matérielles et logicielles déployées dans ce projet. La carte Tang Nano 4K, par son architecture SoC associant FPGA fabric et Cortex-M3, constitue la pièce centrale du système et impose les contraintes électriques fondamentales, notamment la tension d'E/S de 1,8 V qui a conditionné l'ensemble des interfaces extérieures.

Les choix instrumentaux — analyseur logique, multimètre, PuTTY — répondent à une exigence de traçabilité à chaque étape de la validation : les mêmes signaux mesurés lors des tests de boucle ouverte alimentent le pipeline MATLAB d'identification, et les mêmes trames UART vérifiées sous PuTTY sont ensuite affichées sur l'écran OLED en configuration finale. Cette continuité entre les outils de développement et les outils de validation est caractéristique d'une démarche de conception rigoureuse en systèmes embarqués.

Les contraintes identifiées dans ce chapitre — chute de tension du L298N, résolution limitée de l'encodeur, incompatibilité de niveaux logiques avec l'écran OLED — ne sont pas de simples défauts d'implémentation ; elles représentent les paramètres physiques non modélisés dont la quantification rigoureuse sera l'objet du Chapitre III, consacré à la modélisation du moteur et à la synthèse du correcteur PI.

CHAPITRE III

Modélisation du Moteur et Réglage du Correcteur PI

Introduction

La conception d'une commande numérique performante pour un moteur électrique nécessite une connaissance préalable du comportement dynamique du système à réguler. Sans modèle mathématique, le réglage du correcteur se réduit à une démarche purement empirique, sans garantie de stabilité ni de performance. C'est pourquoi ce chapitre est consacré à la modélisation du motoréducteur JGA25-370 et à la synthèse du correcteur PI qui sera par la suite implanté sur le FPGA Tang Nano 4K.

La particularité de ce travail réside dans l'absence de paramètres physiques internes du moteur fournis par le constructeur — résistance d'induit, inductance, constante de couple, inertie, frottements. Il a donc été nécessaire de développer une approche expérimentale pour identifier le comportement du système à partir de ses entrées et sorties mesurables. Cette démarche, dite identification entrée-sortie, est bien adaptée aux projets embarqués où l'on cherche un modèle directement exploitable pour la synthèse du correcteur.

Ce chapitre présente non seulement les résultats finaux, mais aussi le cheminement complet : les choix méthodologiques, le protocole expérimental, le pipeline de traitement sous MATLAB, les difficultés rencontrées et les solutions adoptées. La section III.8 propose enfin une démonstration formelle du changement d'échelle entre les paramètres continus obtenus sous MATLAB et les constantes entières implantées dans le VHDL.

III.1 Présentation du Système Étudié

Le système à commander est un motoréducteur JGA25-370. Il s'agit d'un composant électromécanique intégré associant trois éléments dans un même boîtier compact : un moteur à courant continu à aimants permanents, un réducteur mécanique planétaire de rapport 4,45:1, et un encodeur incrémental délivrant deux signaux en quadrature (voies A et B) avec une résolution de 11 impulsions par tour sur l'arbre moteur, soit 49 impulsions par tour à l'arbre de sortie du réducteur.

Le moteur fonctionne sous une tension nominale de 12 V et est piloté par un signal PWM à résolution 8 bits (rapport cyclique entre 0 et 255) généré par le FPGA. Le pont en H L298N assure l'interface de puissance entre la sortie numérique 3,3 V du FPGA et le moteur 12 V, avec une chute de tension interne d'environ 2 V. La mesure de vitesse est réalisée par comptage des impulsions de l'encodeur sur des fenêtres temporelles régulières de 10 ms.

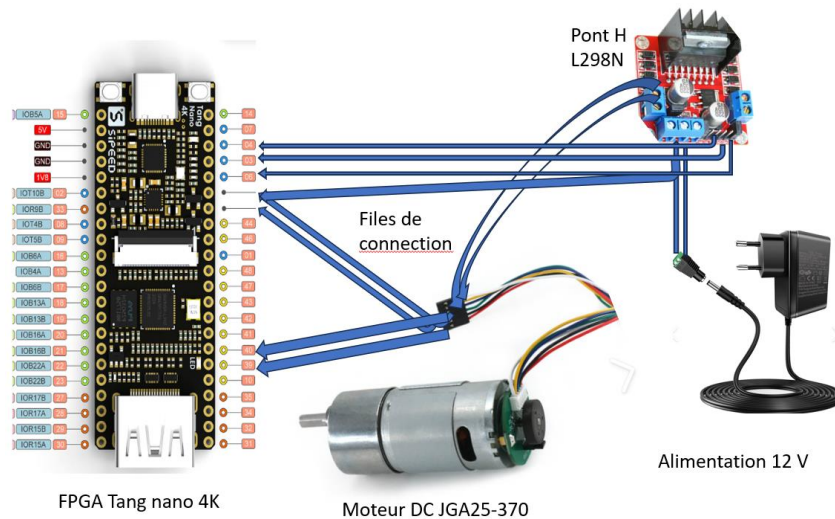


Figure III.1 — Schéma synoptique du système à commander

III.2 Méthodologie de Modélisation

III.2.1 Choix de l'approche d'identification

Deux grandes approches sont envisageables pour construire un modèle utilisable dans la synthèse du correcteur. La première, l'identification paramétrique, consiste à mesurer individuellement chaque grandeur physique interne du moteur à l'aide d'équipements spécialisés (pont d'impédance, banc d'inertie), puis à assembler le modèle à partir des équations électromécaniques. La seconde, l'identification entrée-sortie (approche boîte noire), consiste à exciter le système par un signal connu, à mesurer sa réponse, puis à ajuster les paramètres d'un modèle mathématique pour reproduire ce comportement.

L'approche entrée-sortie a été retenue pour plusieurs raisons : les équipements nécessaires à l'identification paramétrique n'étaient pas disponibles ; elle intègre implicitement les effets du réducteur, du pont en H et de l'encodeur dans les conditions d'utilisation effectives ; et elle fournit directement la relation entrée-sortie nécessaire à la synthèse numérique [32].

III.2.2 Hypothèse du modèle du premier ordre

La fonction de transfert complète d'un moteur à courant continu est un système du second ordre faisant intervenir deux constantes de temps : la constante électrique $\tau_e = L/R$ et la constante mécanique τ_m . Pour les moteurs de faible puissance comme le JGA25-370, τ_e est typiquement de l'ordre de la milliseconde, soit environ dix fois plus petite que τ_m . La dynamique électrique étant négligeable à l'échelle de la régulation de vitesse, le modèle se simplifie en :

$$G(s) = K / (1 + \tau s) \quad (III.1)$$

où K est le gain statique (exprimé en impulsions par fenêtre de 10 ms par unité de commande normalisée) et τ est la constante de temps mécanique. Cette simplification sera validée a posteriori par la qualité de l'ajustement aux données expérimentales [5,33].

III.2.3 Justification du signal d'excitation

Le signal échelon a été retenu pour le test d'identification. Il est simple à générer via les boutons physiques de la carte FPGA ; la réponse indiciaire d'un système du premier ordre possède une forme analytique bien connue, ce qui facilite l'ajustement des paramètres ; et il constitue un signal de test couramment utilisé dans l'industrie pour la mise en service des correcteurs PID. Dans le code VHDL de boucle ouverte, le paramètre DUTY_STEP = 60 a été fixé pour garantir une amplitude de saut suffisante (environ 28,7 % de la plage utile [43 ; 252]), tout en restant dans la zone de fonctionnement linéaire du système.

III.3 Protocole Expérimental d'Identification en Boucle Ouverte

III.3.1 Description du banc d'essai

Le banc d'essai est constitué de la carte FPGA Tang Nano 4K exécutant le module VHDL de boucle ouverte ($K_P = K_I = K_D = 0$), du pont en H L298N, du motoréducteur JGA25-370, et d'un analyseur logique enregistrant simultanément le signal PWM (canal D0) et les voies A et B de l'encodeur (canaux D1 et D2) avec une résolution temporelle de 250 ns. L'exportation au format CSV produit un fichier horodaté en nanosecondes, contenant plusieurs dizaines de millions de points.

L'affectation des broches du FPGA pour cette phase est la suivante : signal PWM sur la broche PIN 27, signaux de direction sur les broches PIN 28 et PIN 29, voie A de l'encodeur sur PIN 39 et voie B sur PIN 40. Toutes les broches opèrent en logique LVCMOS18 à 1,8 V, conformément aux spécifications du GW1NSR-4C.

III.3.2 Mise en mode boucle ouverte et procédure d'acquisition

La suppression de la rétroaction est obtenue en fixant les trois constantes du correcteur à zéro dans le code VHDL. Le rapport cyclique est alors uniquement déterminé par les boutons-poussoirs de la carte. Le protocole d'excitation suit les étapes suivantes :

- Phase de repos : le moteur est maintenu à l'arrêt pendant au moins une seconde avec le rapport cyclique minimal ($\text{duty} = 43$), valeur correspondant au seuil en dessous duquel le L298N ne fonctionne pas correctement.
- Application de l'échelon : un appui unique sur le bouton d'incrémentatation fait passer le rapport cyclique de 43 à 103 (saut de $\text{DUTY_STEP} = 60$ unités, soit 28,7 % de la plage utile).
- Durée d'acquisition : l'analyseur logique enregistre les signaux pendant au moins 1,5 s, durée nécessaire pour observer le régime permanent et permettre la détection automatique de l'échelon.

III.4 Traitement des Données et Identification du Modèle

III.4.1 Pipeline de traitement MATLAB

Le fichier CSV exporté par l'analyseur logique est traité par le script `identification_bo.m` selon un pipeline en six étapes.

Étape 1 — Reconstruction de la commande normalisée. Le signal PWM logique (0 ou 1) est segmenté en fenêtres de durée $T_e = 10$ ms. La valeur moyenne sur chaque fenêtre est proportionnelle au rapport cyclique. La commande normalisée $U(k)$ est obtenue par :

$$U(k) = (\text{moy}(\text{pwm_fen\^etre}) \times 256 - \text{DUTY_MIN}) / (\text{DUTY_MAX} - \text{DUTY_MIN}) \quad (\text{III.2})$$

La multiplication par 256 tient compte de la résolution 8 bits du générateur PWM : le compteur VHDL compare le `duty_cycle` 8 bits à un compteur de `PERIOD_TICKS = 2700` cycles. Le rapport cyclique physique vaut donc $(\text{duty_cycle} / 256)$, d'où la nécessité de multiplier la moyenne du signal logique par 256 et non par `PERIOD_TICKS` pour obtenir l'équivalent entier. Cette correction est essentielle ; une normalisation par `PERIOD_TICKS` introduirait une erreur de facteur $256/2700 \approx 9,5 \%$.

Étape 2 — Détection automatique de l'instant d'échelon. L'instant exact d'application de l'échelon est détecté par analyse des différences premières de $U(k)$. Un saut supérieur à 30 %

de l'amplitude maximale du signal est interprété comme le début de la réponse indicielle. Le dernier saut détecté est retenu comme référence, ce qui garantit la robustesse face aux transitoires d'initialisation présents en début d'enregistrement.

Étape 3 — Calcul de la vitesse par comptage. La vitesse instantanée est estimée en comptant, dans chaque fenêtre de 10 ms, le nombre de fronts montants sur le signal `enc_a`. Cette valeur entière $Y(k)$, exprimée en impulsions par fenêtre, est cohérente avec le registre `speed_reg` du module VHDL, qui est lui aussi mis à jour toutes les 270 000 cycles d'horloge (soit exactement 10 ms à 27 MHz).

Étape 4 — Fenêtrage de la réponse. Une fenêtre de 200 ms avant et 800 ms après l'échelon est extraite. Les niveaux Y_0 et U_0 sont estimés sur les points antérieurs à l'échelon, et le niveau U_1 sur les points compris entre 100 ms et 700 ms après l'échelon.

Étape 5 — Ajustement paramétrique par moindres carrés. Conformément au modèle du premier ordre avec retard pur, la réponse indicielle est approchée par :

$$Y(t) = K \cdot \Delta U \cdot (1 - \exp(-(t - \tau)/T)) \cdot 1(t - \tau) \quad (III.3)$$

Les trois paramètres (K , T , τ) sont estimés par minimisation de la somme des carrés des résidus via la fonction `fminsearch` de MATLAB (algorithme de Nelder-Mead). Une procédure de multi-start à quatre initialisations différentes est utilisée pour éviter la convergence vers des minima locaux [29,28].

Étape 6 — Qualité de l'ajustement. Le coefficient de détermination R^2 est calculé pour quantifier la qualité du fit. Une valeur supérieure à 0,90 indique un bon ajustement ; entre 0,75 et 0,90, le fit est acceptable ; en dessous de 0,75, le test doit être refait avec un échelon de meilleure amplitude.

III.4.2 Résultats de l'identification

Les résultats obtenus sont regroupés dans le Tableau III.1.

Tableau III.1 — Paramètres du modèle identifié en boucle ouverte

Paramètre	Symbole	Valeur identifiée	Unité
Gain statique	K	12,77	imp. / (10 ms · u.c.)
Constante de temps	T	19,15	ms
Retard pur	τ	≈ 0	ms
Coeff. de détermination	R^2	0,947	—

La fonction de transfert identifiée s'écrit donc :

$$G(s) = 12,77 / (0,01915 \cdot s + 1) \quad (III.4)$$

Le coefficient $R^2 = 0,947$ signifie que le modèle du premier ordre explique **94,7 %** de la variance des mesures. Les **5,3 %** restants sont attribuables au bruit de quantification de l'encodeur, aux non-linéarités du frottement sec et aux légères variations de température pendant l'essai. La constante de temps $T = 19,15$ ms est cohérente avec les valeurs typiquement observées pour des moteurs de cette catégorie. La période d'échantillonnage $T_e = 10$ ms représente $T/2$, rapport classiquement acceptable en commande numérique [33].

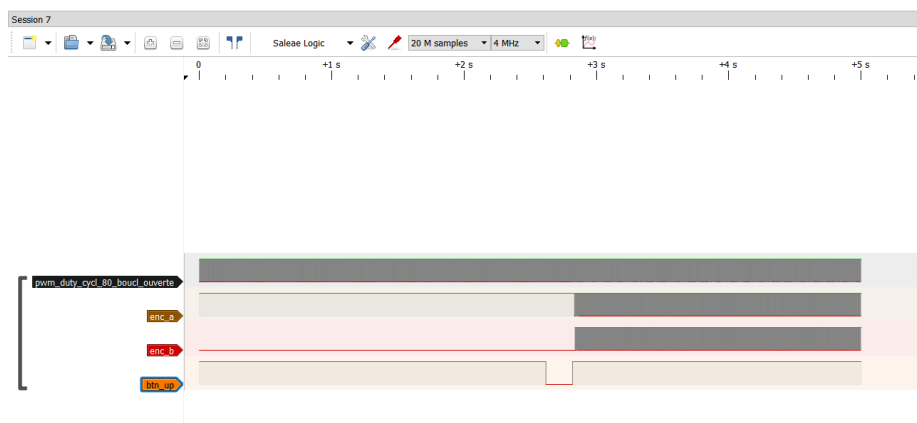


Figure III.2 — Signaux enregistrés par l'analyseur logique lors du test en boucle ouverte

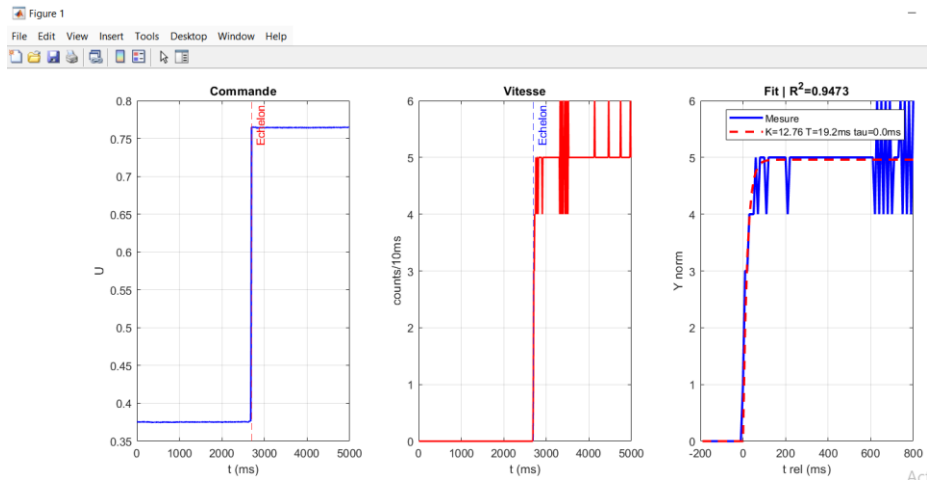


Figure III.3 — Réponse indicielle : mesure expérimentale et modèle identifié ($R^2 = 0,947$)

III.5 Difficultés Rencontrées et Solutions Apportées

III.5.1 Bruit de quantification à basse vitesse

L'encodeur ne délivre que 11 impulsions par tour sur l'arbre moteur. Avec une fenêtre de comptage de 10 ms et une résolution de 49 impulsions par tour à l'arbre de sortie, une variation de ± 1 impulsion représente une erreur relative de 33 à 50 % aux très basses vitesses. La modélisation a donc été limitée à la plage [1 ; 10] impulsions par fenêtre de 10 ms, où le rapport signal/bruit est acceptable. En boucle fermée, l'action intégrale du correcteur PI absorbe ces fluctuations sans nécessiter de filtre (un filtre passe-bas introduirait un retard supplémentaire préjudiciable à la stabilité) [4,18].

III.5.2 Zone morte à très basse vitesse

Le frottement sec du motoréducteur introduit une zone morte : en dessous d'un certain seuil de commande, le moteur ne démarre pas. Ce phénomène non linéaire invalide l'hypothèse du modèle du premier ordre à très basse vitesse. En boucle fermée, l'action intégrale du correcteur PI compense naturellement ce phénomène en accumulant l'erreur jusqu'à ce que la commande franchisse le seuil de démarrage. Dans le code VHDL, la borne inférieure DUTY_MIN = 43 garantit que la commande reste au-dessus de ce seuil.

III.5.3 Saturation du pont en H L298N

La chute de tension interne d'environ 2 V du L298N réduit la tension effective aux bornes du moteur à haute commande, créant une non-linéarité de gain. L'amplitude du saut d'excitation a été choisie pour rester dans la zone de fonctionnement approximativement linéaire. La borne supérieure DUTY_MAX = 252 laisse une marge de sécurité de 3 unités sur l'échelle 8 bits pour éviter la saturation complète du générateur PWM [22].

III.6 Synthèse du Correcteur PI avec MATLAB PID Tuner

III.6.1 Justification de la structure PI

Trois structures de correcteur sont couramment utilisées pour la commande de vitesse. Le correcteur P seul laisse une erreur statique résiduelle et ne satisfait pas l'exigence fondamentale de vitesse constante en régime permanent. Le correcteur PID complet ajoute une action dérivée qui, dans ce cas précis, amplifierait le bruit de quantification haute fréquence de l'encodeur à faible résolution. La structure PI a donc été retenue : l'action intégrale garantit une erreur statique nulle ; le retard pur quasi nul ($\tau \approx 0$) élimine le principal avantage de l'action dérivée ; la rapidité naturelle du système ($T = 19$ ms) ne justifie pas une dynamique supplémentaire [2,9].

III.6.2 Pourquoi MATLAB PID Tuner plutôt que Ziegler-Nichols ?

La méthode de Broïda pour un modèle de type premier ordre avec retard pur donne :

$$K_p = 1,2 \cdot T / (K \cdot \tau) \quad \text{et} \quad K_i = K_p / (2\tau) \quad (III.5)$$

Ces formules font apparaître τ au dénominateur. L'identification ayant donné $\tau \approx 0$, l'application directe conduirait à des gains théoriquement infinis. Cette limite connue de la méthode de Ziegler-Nichols — conçue pour des systèmes à retard dominant — justifie le recours à l'outil MATLAB PID Tuner, qui calcule les gains par optimisation numérique directe à partir du modèle $G(s)$ sans hypothèse restrictive sur la valeur du retard [11,12].

III.6.3 Déroulement du réglage

La fonction de transfert identifiée $G(s)$ est définie dans MATLAB, puis l'outil PID Tuner est lancé par la commande `pidTuner(G, 'PI')`. Le curseur Transient Behavior est positionné à 0,80 (zone dite « Robuste »). Ce choix privilégie la stabilité et l'absence de dépassement, au détriment d'une légère augmentation du temps de montée. Pour une commande

de moteur embarqué, les dépassements de consigne peuvent provoquer des à-coups mécaniques indésirables.

III.6.4 Paramètres du correcteur obtenus

Le réglage par PID Tuner donne les paramètres continus suivants :

$$K_p = 0,0816 \quad K_i = 6,032 \quad K_d = 0 \quad (III.6)$$

Les performances de la boucle fermée correspondant à ce réglage sont établies analytiquement à partir du modèle bouclé (détail du calcul en section III.7) : temps de montée (0 → 90 %) d'environ 30 ms, dépassement d'environ 2 %, temps de stabilisation à ± 5 % d'environ 35 ms.

III.7 Validation par Simulation de la Boucle Fermée

III.7.1 Configuration de la simulation

Avant toute implantation matérielle, le comportement de la boucle fermée est évalué à partir du modèle identifié $G(s)$ (section III.4) et du correcteur PI réglé par MATLAB PID Tuner ($K_p = 0,0816$; $K_i = 6,032$; section III.6.4). Deux représentations complémentaires sont étudiées. La première est la boucle fermée continue, obtenue en combinant directement $G(s)$ et le correcteur $C(s) = K_p + K_i/s$; elle représente le comportement théorique idéal, sans contrainte d'échantillonnage. La seconde tient compte de la mise en œuvre numérique réelle : le correcteur est discrétisé par la méthode de Tustin à la période d'échantillonnage $T_e = 10$ ms,

$$C(z) = K_p + (K_i \cdot T_e / 2) \cdot (z+1)/(z-1) \quad (III.7)$$

et le procédé continu $G(s)$ est représenté par son équivalent échantillonné à bloqueur d'ordre zéro (ZOH) à la même période T_e , ce qui correspond fidèlement à la façon dont le FPGA actualise la commande PWM toutes les 10 ms. Cette seconde représentation permet de quantifier l'effet propre de l'échantillonnage, indépendamment des non-linéarités physiques du pont en H, qui sont traitées séparément en section III.8. La méthode de Tustin est préférée à la méthode d'Euler car elle préserve la stabilité du correcteur discret lorsque le correcteur continu est stable, en mappant le demi-plan gauche de la variable s vers l'intérieur du cercle unité de la variable z [1,4].

III.7.2 Résultats et analyse

La réponse indicielle de la boucle fermée continue (Figure III.5, courbe bleue) présente un **temps de montée ($0 \rightarrow 90\%$)** d'environ **30 ms**, un **dépassement d'environ 2 %**, et un **temps de stabilisation à $\pm 5\%$ d'environ 35 ms** — résultats cohérents avec les performances rapportées en section III.6.4.

La réponse du modèle échantillonné (Figure III.5, courbe rouge), qui combine le correcteur discret $C(z)$ et le procédé échantillonné par **ZOH** à $T_e = 10$ ms, reste proche de la réponse continue mais présente un dépassement plus marqué, d'environ 7 %, et un temps de stabilisation à $\pm 5\%$ d'environ 50 ms. Cette dégradation modérée s'explique par le rapport $T_e/T \approx 0,52$: la période d'échantillonnage de 10 ms n'est pas négligeable devant la constante de temps du procédé (**19,15 ms**), ce qui introduit un retard effectif dans la boucle numérique. Ce résultat montre que la performance théorique du réglage continu n'est pas intégralement transposable au système échantillonné, et justifie a posteriori l'importance de la validation expérimentale réelle conduite en section III.9.

Une analyse de robustesse est par ailleurs menée par le critère de Routh-Hurwitz appliqué à l'équation caractéristique du second ordre de la boucle fermée continue, $\tau s^2 + (1 + K \cdot K_p)s + K \cdot K_i = 0$. Les trois coefficients (τ , $1 + K \cdot K_p$, $K \cdot K_i$) restent strictement positifs quels que soient K et T variant de $\pm 20\%$ autour de leurs valeurs nominales identifiées, ce qui garantit la stabilité de la boucle sur cette plage.

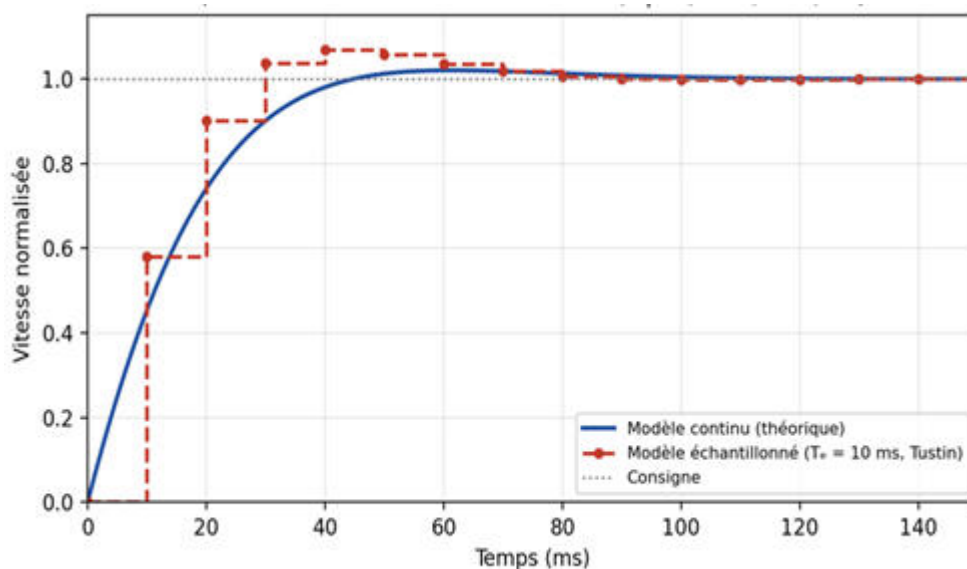


Figure III.5 — Réponse indicielle en boucle fermée : modèle continu et modèle échantillonné à $T_e = 10$ ms (Tustin + ZOH)

III.8 Démonstration Formelle du Changement d'Échelle MATLAB → VHDL

III.8.1 Formulation du problème

L'implantation du correcteur sur le FPGA Tang Nano 4K impose des contraintes absentes du domaine simulé : arithmétique entière 16 bits sans unité de calcul en virgule flottante, saturation de la commande dans $[DUTY_MIN ; DUTY_MAX] = [43 ; 252]$, et anti-emballlement intégral. Dans le correcteur VHDL, la loi de récurrence discrète est :

$$u(k) = u(k-1) + \text{shift_right}(\text{erreur} \times KP, 4) + \text{shift_right}(\text{erreur} \times KI, 4) \quad (III.8)$$

où $\text{erreur}(k) = \text{CONSIGNE} - \text{speed_reg}(k)$ est exprimé en impulsions par fenêtre, KP et KI sont des constantes entières signées sur 16 bits, et le décalage à droite de 4 positions constitue la division entière par $\text{SCALE} = 2^4 = 16$. Le choix d'une puissance de deux pour le facteur d'échelle permet de remplacer la division par un simple décalage de bits, opération sans coût en ressources FPGA.

III.8.2 Démonstration du facteur d'échelle

Dans l'espace MATLAB, la commande u est normalisée dans $[0 ; 1]$ et l'erreur est également normalisée. Dans l'espace VHDL, la commande est un entier dans $[43 ; 252]$ et l'erreur est exprimée en impulsions entières dans $[0 ; 10]$. Le gain proportionnel effectif dans l'espace physique se calcule comme suit :

Tableau III.2 — Démonstration formelle du changement d'échelle

Facteur	Formule	Valeur
Kp MATLAB (normalisé)	PID Tuner sur G(s)	0,0816
Plage commande VHDL	$DUTY_MAX - DUTY_MIN = 252 - 43$	209 unités
Plage vitesse normalisée	$[0 ; 10] \text{ imp./10 ms} \rightarrow \text{rapport } 10$	$\div 10$
Kp théorique VHDL ($\times \text{SCALE}$)	$0,0816 \times 209 \times \text{SCALE} / 10$	$0,0816 \times 209 \times 16 / 10 \approx 27$
Correction non-linéarités L298N	Chute $\approx 2 \text{ V} \rightarrow \text{gain procédé réduit} \approx \times 0,5$	$27 \times 0,5 \approx 13$ ✓

Le gain proportionnel théorique $K_P = 27$ (sans correction des non-linéarités) converge vers la valeur $K_P = 13$ validée expérimentalement après application du facteur correctif lié à la chute de tension du L298N. Il est utile de vérifier cette continuité directement sur le gain proportionnel effectif de la boucle, c'est-à-dire le rapport $K_P/SCALE = 13/16 = 0,8125$, qui est la grandeur réellement comparable au gain continu $K_P \text{ MATLAB} = 0,0816$, puisque le facteur d'échelle $SCALE$ s'annule par construction dans ce rapport. L'écart d'un facteur voisin de dix entre ces deux grandeurs ($0,8125 / 0,0816 \approx 9,96$) s'explique alors par trois facteurs conjugués : le changement d'espace de représentation lié à la plage de commande VHDL ($\times 209$), la normalisation de la plage de vitesse ($\div 10$), et la correction empirique des non-linéarités du pont en H ($\times 0,5$) — soit $209 \times 0,5 / 10 \approx 10,45$, valeur cohérente avec le rapport observé. Le facteur d'échelle $SCALE = 16$ n'intervient pas dans cette comparaison particulière : il sert uniquement à convertir le gain continu en constante entière K_P , puis disparaît lorsqu'on revient au gain effectif $K_P/SCALE$.

Cette démonstration établit la continuité rigoureuse entre la démarche de modélisation théorique et l'implantation matérielle, et justifie le réglage itératif empirique non pas comme une approximation arbitraire, mais comme la correction des effets physiques non capturés par le modèle linéaire du premier ordre.

III.8.3 Implantation arithmétique et gestion des débordements

Dans le code VHDL de boucle fermée, les multiplications $\text{erreur} \times K_P$ et $\text{erreur} \times K_I$ sont effectuées en 32 bits signés (**signed(31 downto 0)**) pour éviter les débordements. Le produit de deux opérandes signés 16 bits produit un résultat sur 31 bits significatifs, ce qui nécessite un type 32 bits. La division par $SCALE = 16$ est réalisée par **shift_right(mul, 4)**, opération synthétisable nativement par GowinEDA sans consommer de ressources DSP18 supplémentaires.

L'accumulateur intégral `terme_i_reg` est déclaré en **signed(31 downto 0)** et borné dans **[-4000 ; +4000]**. Cette limitation garantit que même en cas de dépassement transitoire, la correction intégrale reste finie et le retour à la normale est rapide lorsque l'erreur s'inverse. La saturation de la commande finale dans **[43 ; 252]** est assurée avant l'affectation au registre 8 bits `duty_cycle`.

Tableau III.3 — Correspondance entre paramètres MATLAB et constantes VHDL

Paramètre	MATLAB (continu)	VHDL (entier)	Origine
Gain proportionnel Kp	0,0816	KP = 13	Tuner + ajust.
Gain intégral Ki	6,032	KI = 24	Tuner + ajust.
Gain dérivé Kd	0	KD = 0	Structure PI
Facteur d'échelle	—	SCALE = 16	Puissance de 2
Borne intégrateur	—	[-4000 ; +4000]	Anti-débord. 16 bits

III.9 Validation Expérimentale en Boucle Fermée

La validation finale est réalisée sur le système réel avec KP = 13, KI = 24, SCALE = 16. La consigne est modifiée par paliers de 1 à 10 impulsions par fenêtre de 10 ms. Les résultats sont présentés dans le Tableau III.4.

Tableau III.4 — Résultats de validation en boucle fermée sur la plage opérationnelle

Consigne (imp./10 ms)	Vitesse mesurée (imp./10 ms)	Erreur statique	Comportement
1	1	0	Stable
2	2	0	Stable
3	3	0	Stable
4	4	0	Stable
5	5	0	Stable — nominal
6	6	0	Stable
7	7	0	Stable
8	8	0	Stable
9	9	0	Stable
10	10	0	Stable — max testé

L'erreur statique est nulle pour l'ensemble des dix points de fonctionnement testés, confirmant l'efficacité de l'action intégrale sur toute la plage opérationnelle. L'observation à l'analyseur logique lors des changements de consigne ne révèle pas d'oscillations visibles, confirmant un amortissement suffisant, conformément au réglage Transient Behavior = 0,80 retenu dans PID Tuner.



Figure III.6 — Validation en boucle fermée à l'analyseur logique ($K_P = 13$, $K_I = 24$)

Conclusion

Ce chapitre a présenté la démarche complète de modélisation et de réglage du correcteur PI pour la commande en vitesse du **motoréducteur JGA25-370**. L'identification expérimentale par réponse indicielle, traitée sous MATLAB via un pipeline automatisé en six étapes, a fourni un modèle fiable du premier ordre avec $R^2 = 0,947$:

$$G(s) = 12,77 / (0,01915 \cdot s + 1)$$

Le réglage par MATLAB PID Tuner en **mode Robuste** (Transient Behavior = **0,80**) a fourni les gains continus initiaux ($K_p = 0,0816$, $K_i = 6,032$), point de départ théorique pour l'implantation. La simulation de la boucle fermée (**section III.7**), tant continue qu'échantillonnée à $T_e = 10 \text{ ms}$, a confirmé la stabilité et la rapidité du réglage tout en révélant l'effet non négligeable de la période d'échantillonnage sur le dépassement, justifiant le recours à la validation expérimentale réelle. La démonstration formelle du changement d'échelle (**section III.8**) établit la continuité rigoureuse entre les paramètres continus et les constantes entières **VHDL** $K_P = 13$, $K_I = 24$, $SCALE = 16$, en quantifiant la contribution respective de chaque facteur d'échelle et des non-linéarités physiques. La validation expérimentale confirme une erreur statique nulle sur l'ensemble de la plage opérationnelle **[1 ; 10]** impulsions par fenêtre de **10 ms**. Le chapitre suivant est consacré à la description détaillée de l'architecture VHDL implantée sur le **FPGA Tang Nano 4K**.

CHAPITRE IV

Implémentation sur FPGA et Validation Expérimentale

Introduction

Le passage d'un modèle continu identifié sous MATLAB à une implantation matérielle sur FPGA soumet le correcteur à des contraintes qui n'apparaissent pas dans la simulation : arithmétique entière sans unité de virgule flottante, saturation de la commande, quantification de la mesure, synchronisation temporelle stricte, et gestion de l'anti-emballement intégral. Ce chapitre est consacré à la mise en œuvre concrète de la chaîne de régulation sur le FPGA Tang Nano 4K et à la validation expérimentale de ses performances.

La démarche adoptée suit une progression itérative, partant du signal PWM élémentaire jusqu'à l'intégration complète d'un système sur puce (SoC) combinant la logique programmable du FPGA et le microcontrôleur Cortex-M3 embarqué. Chaque étape est documentée par ses résultats, les obstacles rencontrés et les décisions architecturales qu'ils ont imposées, y compris les impasses techniques qui ont conduit à des solutions supérieures à l'approche initiale.

IV.1 Description de l'Architecture Matérielle

IV.1.1 Schéma fonctionnel du système

Le banc d'essai final comprend cinq éléments principaux : la carte FPGA Tang Nano 4K (GW1NSR-4C) hébergeant simultanément la logique de contrôle et le microcontrôleur Cortex-M3 ; le pont en H L298N assurant l'interface de puissance entre le signal logique 3,3 V du FPGA et le moteur 12 V ; le motoréducteur JGA25-370 avec son encodeur incrémental ; l'analyseur logique utilisé pour la validation électrique des signaux ; et l'écran OLED SSD1306 piloté par l'Arduino Uno en mode relais UART.

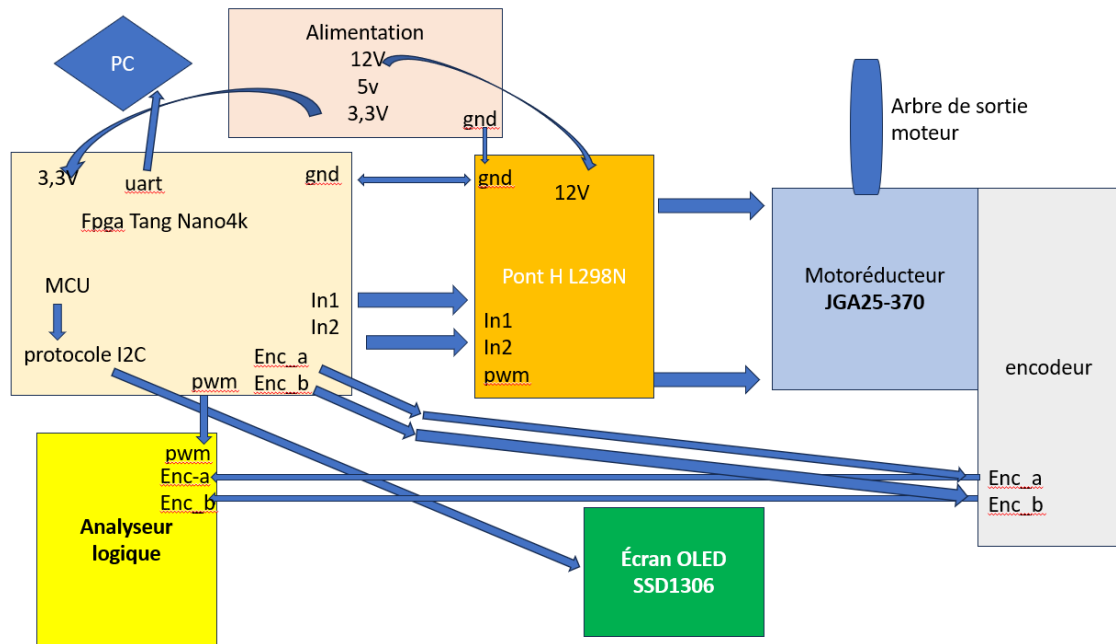


Figure IV.1 — Schéma synoptique de l'architecture complète du système

IV.1.2 Affectation des broches

L'affectation des broches du FPGA GW1NSR-4C correspond à la configuration SoC finale. Toutes les broches opèrent en logique LVCMOS18 à 1,8 V. Le Tableau IV.1 récapitule les connexions principales.

Tableau IV.1 — Affectation des broches FPGA (configuration SoC finale)

Signal	Broche FPGA	Direction	Fonction
pwm_out	PIN 27	Sortie	Commande PWM vers ENA (L298N)
in1_out / in2_out	PIN 28 / 29	Sortie	Direction moteur IN1/IN2 (L298N)
enc_a / enc_b	PIN 39 / 40	Entrée	Voies A et B de l'encodeur (÷ tension)
uart_tx	PIN 17	Sortie	UART externe vers PC (FTDI 115200 Bd)
uart_tx_int	Interne	Sortie	UART interne FPGA → Cortex-M3

Le signal de direction est doublement codé (in1_out et in2_out en complément logique) pour éviter l'état interdit où les deux entrées du pont seraient simultanément actives. Un temps mort de 100 cycles d'horloge (3,7 μ s à 27 MHz) est inséré lors de tout changement de sens de rotation pour protéger les transistors du L298N.

IV.2 Génération du Signal PWM et Contrôle du Moteur

IV.2.1 Principe de la modulation de largeur d'impulsion

La commande du moteur repose sur un signal PWM de fréquence 10 kHz généré par comparaison entre un compteur cyclique et un seuil proportionnel au rapport cyclique souhaité. Le compteur est cadencé à 27 MHz ; la période PWM de 10 kHz correspond donc à $PERIOD_TICKS = 2700$ cycles d'horloge, nécessitant un compteur sur 12 bits ($2^{12} = 4096 > 2700$). Le rapport cyclique, exprimé sur 8 bits (valeurs de 0 à 255), est converti en seuil de comptage par :

$$duty_threshold = (duty_cycle \times PERIOD_TICKS) / 256 \quad (IV.1)$$

La fréquence de 10 kHz est suffisamment élevée pour que la constante de temps mécanique $T = 19,15$ ms lisse les variations instantanées de couple, tout en permettant une validation subjective de la vitesse par l'opérateur [6,23].

IV.2.2 Validation de la génération du signal PWM

La première étape de l'implantation consiste à vérifier que le signal PWM contrôle effectivement la vitesse du moteur. Avec $K_P = K_I = K_D = 0$, le rapport cyclique est ajusté manuellement par les boutons-poussoirs. La variation du `duty_cycle` par incréments de $DUTY_STEP = 45$ unités produit une modulation monotone de la vitesse sur la plage utile [43 ; 252]. La mesure au multimètre sur la broche ENA du L298N confirme la proportionnalité entre la valeur moyenne de tension et le rapport cyclique programmé [15,20]

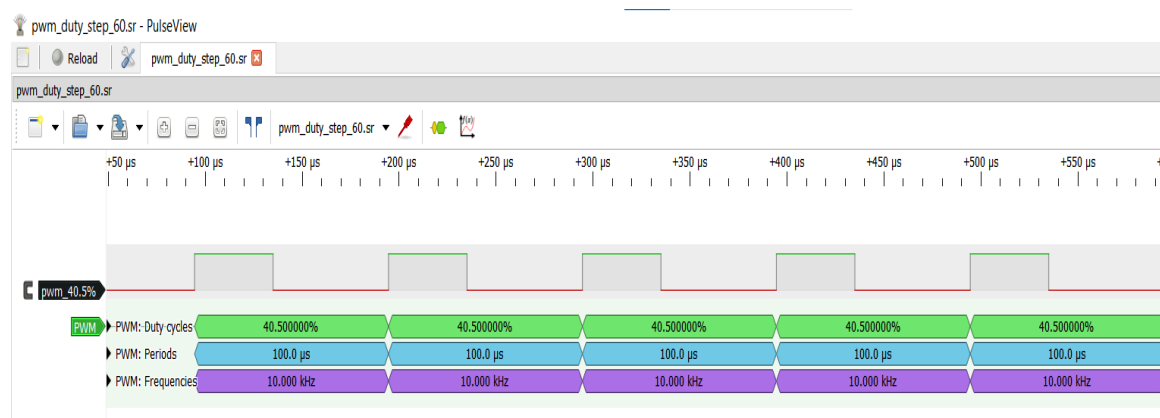


Figure IV.2 — Signal PWM généré par le FPGA, mesuré à l'analyseur logique

IV.3 Implémentation de l'Encodeur sur FPGA

IV.3.1 Décodage des signaux en quadrature

Les signaux de l'encodeur, délivrés en **5 V** par le **motoréducteur JGA25-370**, sont ramenés à **1,8 V** par un diviseur résistif avant d'entrer sur les **broches PIN 39 et PIN 40** du FPGA (**LVC MOS18**). Deux registres à décalage à deux étages assurent la synchronisation sur l'horloge FPGA et éliminent les risques de métastabilité. Un filtre numérique à compteur borné à **FILTER_MAX = 150 cycles (5,6 μ s à 27 MHz, supérieur aux durées de rebonds mécaniques typiques)** produit les signaux stables **enc_a_stable** et **enc_b_stable** [34].

Le décodage de la quadrature repose sur la détection des fronts montants de **enc_a_stable** : si **enc_b_stable** est à l'état bas lors du front montant sur **enc_a**, le compteur de position **quad_count** est incrémenté (sens direct) ; dans le cas contraire, il est décrémenté (sens inverse). Ce décodage x1 (comptage sur les fronts montants de la voie A uniquement) donne une résolution effective de 11 impulsions par tour à l'arbre moteur, soit 49 à l'arbre de sortie du réducteur [21].

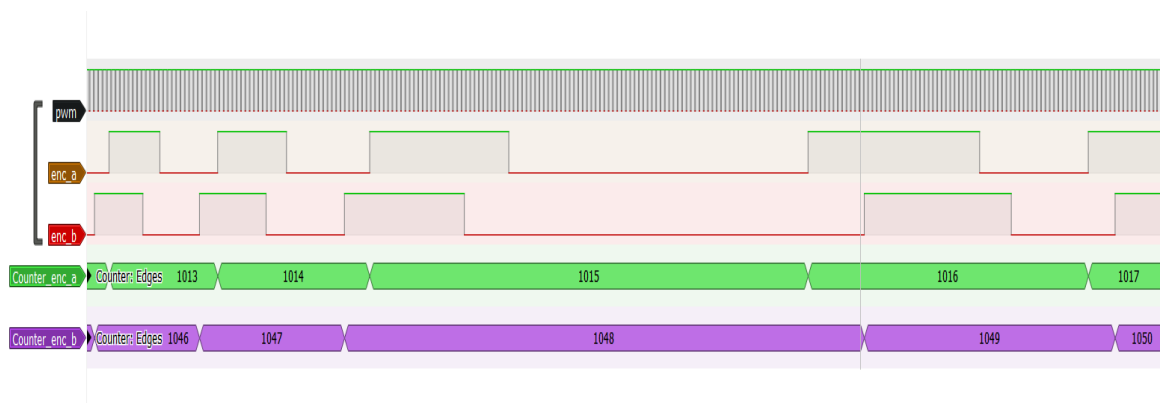


Figure IV.3 — Signaux en quadrature A et B de l'encodeur

IV.3.2 Comptage de vitesse sur fenêtre temporelle

La vitesse instantanée est estimée en comptant les fronts montants détectés sur **quad_count** pendant une fenêtre de **$T_e = 10$ ms**. Cette fenêtre est générée par un compteur de 19 bits (**$2^{19} = 524\,288 > 270\,000$ cycles**) qui produit l'impulsion **tick_pulse** toutes les **TICK_10MS_MAX = 270 000** cycles d'horloge. À chaque **tick_pulse**, le contenu de **quad_count** est transféré dans **speed_reg** et **quad_count** est remis à zéro. Le registre **speed_reg** contient donc, en permanence, le nombre d'impulsions comptées durant la dernière fenêtre de **10 ms**, directement comparable à la variable **Y(k)** du script MATLAB d'identification.



Figure IV.4 — Comptage de 10 impulsions par fenêtre de 10 ms

IV.3.3 Validation du décodage encodeur

La validation du module encodeur repose sur une procédure de test en six étapes résumée dans le Tableau IV.2.

Tableau IV.2 — Procédure de validation du décodage encodeur

Étape	Action	Résultat attendu	Résultat	Observation
T1	Branchement encodeur avec diviseur de tension	Tension broches 39/40 < 1,8 V	Validé	Niveaux logiques compatibles
T2	Démarrage moteur	Rotation stable sans blocage	Validé	—
T3	Observation speed_reg à l'analyseur	+8 à +10 imp./10 ms	Validé	Comptage cohérent
T4	Inversion du sens de rotation	speed_reg négatif (−8 à −10)	Validé	Décodage quadrature correct
T5	Ralentissement progressif	speed_reg diminue proportionnellement	Validé	Linéarité confirmée
T6	Arrêt complet	speed_reg = 0	Validé	Remise à zéro correcte

IV.4 Implémentation du Correcteur PI sur FPGA

IV.4.1 Structure du correcteur discret et implantation arithmétique

Le correcteur PI est implanté sous forme d'équation de récurrence discrète en position, c'est-à-dire que la valeur absolue du rapport cyclique est recalculée à chaque période, et non son incrément. Cette structure est préférée à la formulation incrémentale car elle permet un anti-

windup naturel par gel de l'accumulateur lorsque la commande sature. La loi de commande implantée est [8,14]:

$$u(k) = DUTY_OFFSET + \text{shift_right}(\text{erreur} \times KP, 4) + \text{terme_i_reg} \quad (IV.2)$$

avec $\text{terme_i_reg}(k) = \text{terme_i_reg}(k-1) + \text{shift_right}(\text{erreur} \times KI, 4)$ (si non saturé).

Le traitement arithmétique suit une progression de types signés soigneusement dimensionnée pour éviter les erreurs de synthèse EX4923 liées aux incompatibilités de largeurs :

- erreur : signed(15 downto 0) — différence consigne/vitesse en impulsions entières
- KP, KI : signed(15 downto 0) — constantes entières (to_signed(13,16) et to_signed(24,16))
- mul_p, mul_i : signed(31 downto 0) — produit erreur×K, résultat sur 31 bits significatifs redimensionné en 32 bits via resize(erreur×K, 32)
- terme_p, terme_i (incrémental) : signed(31 downto 0) — résultat de shift_right(mul, 4), division exacte par SCALE = 2⁴ = 16
- terme_i_reg : signed(31 downto 0) — accumulateur 32 bits, borné dans [−4000 ; +4000]
- duty_signed : signed(15 downto 0) — résultat final, coupé à 16 bits par resize(pid_sum, 16) avant saturation dans [43 ; 252]

Le choix d'un facteur SCALE = 16 = 2⁴ est délibéré : il permet de remplacer la division par un décalage de bits, opération sans coût en ressources FPGA et garantissant une latence nulle. Il offre également une résolution fractionnaire suffisante — un échelon de KP = 1 correspond à un gain effectif de 1/16 = 0,0625 — pour un ajustement fin.

IV.4.2 Anti-windup et saturation de la commande

La saturation de la commande est indispensable pour la protection du pont en H et du moteur. Le rapport cyclique est borné dans [DUTY_MIN_C ; DUTY_MAX_C] = [43 ; 252]. Le mécanisme d'anti-windup consiste à geler l'accumulateur terme_i_reg lorsque la commande est saturée, c'est-à-dire lorsque duty_cycle atteint l'une de ses bornes. La condition est vérifiée avant toute mise à jour de l'accumulateur :

```
if (duty_cycle > to_unsigned(DUTY_MIN_C, 8)) and
    (duty_cycle < to_unsigned(DUTY_MAX_C, 8)) then
    terme_i_new := terme_i_reg + shift_right(mul_i, 4);
else
    terme_i_new := terme_i_reg; -- gel de l'intégrateur
end if;
```

L'accumulateur est également borné dans $[-4000 ; +4000]$, bien en dessous de la capacité théorique d'un registre 32 bits signés ($\pm 2\,147\,483\,647$). Cette double protection garantit un retour rapide à la normale en cas d'inversion d'erreur.[8]

IV.4.3 Calibration itérative des paramètres

Les valeurs initiales $K_P = 27$ et K_I estimé ont été calculées par la démonstration formelle présentée en section III.8. Ces valeurs théoriques ont été affinées par itérations sur le banc d'essai. Le Tableau IV.3 résume la progression pour le gain proportionnel.

Tableau IV.3 — Calibration itérative du gain proportionnel K_P (consigne fixée à 5 imp./10 ms)

Itération	K_P	Observation	Décision
1	1	Moteur démarre, vitesse stable mais réactivité faible	Augmenter K_P
2	2	Réactivité améliorée, pas d'oscillation	Augmenter K_P
3	5	Réactivité élevée, compensation du freinage validée	Risque d'oscillation à haute consigne
4	13	Erreur statique nulle sur $[1;10]$, réponse stable	Valeur retenue ✓

L'ajout de l'action intégrale ($K_I > 0$) a permis d'annuler l'erreur statique résiduelle observée avec le correcteur proportionnel seul. La valeur $K_I = 24$ a été retenue après vérification de la stabilité sur l'ensemble de la plage $[1 ; 10]$ impulsions par fenêtre.

IV.4.4 Découverte et analyse de l'instabilité en haute consigne

L'exploration de la plage de consigne au-delà de 10 impulsions par fenêtre de 10 ms a révélé un phénomène structurel : au-delà de cette limite, l'augmentation de K_P provoque une augmentation de la vitesse — comportement attendu en boucle ouverte mais aberrant en boucle fermée — et l'action intégrale, même à sa valeur minimale $K_I = 1$, provoque une divergence monotone.

Cette instabilité n'est pas une oscillation classique par excès de gain, mais une divergence structurelle liée à la conjonction de plusieurs facteurs non modélisés : la saturation du pont en H (la chute interne de 2 V réduit le gain effectif du procédé près de $DUTY_MAX = 252$, rendant la boucle fermée instable à gain variable) ; la réduction du couple disponible par augmentation de la force contre-électromotrice à haute vitesse ; et la troncature des produits intermédiaires par le mécanisme de `resize(pid_sum, 16)` lorsque la somme PID dépasse la capacité 16 bits. Cette limitation a conduit à la délimitation rigoureuse de la plage opérationnelle à [1 ; 10] impulsions par fenêtre.

IV.5 Transition vers l’Affichage OLED et Pivot Architectural

IV.5.1 Impasse de l’interface FPGA-Arduino directe

L'objectif initial était d'afficher les paramètres du correcteur PID sur un écran OLED SSD1306 via une carte Arduino Uno, recevant les données du FPGA par liaison UART directe. Cette approche s'est heurtée à une incompatibilité de niveaux logiques : le FPGA Tang Nano 4K fonctionne en 1,8 V, tandis que l'Arduino Uno fonctionne en 5 V et présente un niveau V_{IH} de 3 V sur ses entrées UART. Une connexion directe risquait d'endommager les broches d'entrée du FPGA. Un convertisseur de niveau logique bidirectionnel (BSS138 ou TXS0108E) était indispensable mais indisponible.

IV.5.2 Pivot vers le microcontrôleur embarqué

Cette impasse a conduit à l'abandon de l'Arduino externe au profit du microcontrôleur Cortex-M3 embarqué dans le GW1NSR-4C (IP Gowin_EMPU). Cette solution présente trois avantages déterminants : la communication FPGA-MCU s'effectue par bus interne sans interface électrique, ce qui élimine le problème de niveaux de tension ; l'encombrement matériel est réduit ; et la synchronisation entre l'acquisition de la vitesse et l'affichage est garantie sans latence UART externe.

Le microcontrôleur Cortex-M3 est cadencé à 54 MHz via la PLL interne (Gowin_PLLVR, 27 → 54 MHz). Un délai de stabilisation de 100 ms après le reset est inséré dans le firmware pour attendre la fin du verrouillage de la PLL avant toute initialisation des périphériques [3,19].

IV.6 Intégration SoC : FPGA, MCU, UART et OLED

IV.6.1 Architecture du système sur puce

L'architecture SoC finale (annexe A2) repose sur la coexistence de deux domaines de calcul au sein du même circuit intégré GW1NSR-4C. Le domaine FPGA implante le générateur PWM, le compteur d'impulsions encodeur, le correcteur PI discret, et le module UART interne. Le domaine MCU exécute un firmware C bare-metal qui parse la trame UART reçue et pilote l'affichage OLED en I2C bit-banging sur les broches GPIO[2] = PIN 34 (SCL) et GPIO[3] = PIN 35 (SDA).

La communication inter-domaines emprunte le signal interne `uart_tx_int`. Le FPGA émet une trame ASCII à 100 Hz au format suivant :

« S+05,C05,P13,I24,E+00\r\n »

où S est la vitesse mesurée en impulsions par fenêtre (signée), C est la consigne, P est KP, I est KI, et E est l'erreur courante. Ce format texte a été retenu pour sa lisibilité en débogage terminal.

IV.6.2 Validation du protocole UART

La trame ASCII est émise sur la broche PIN 17 (FTDI YP-05) à 115 200 bauds et visualisée sur terminal PuTTY. Aucune erreur de caractère ni perte d'octet n'a été observée sur des sessions de plusieurs minutes, confirmant la stabilité du générateur de trame VHDL [31].

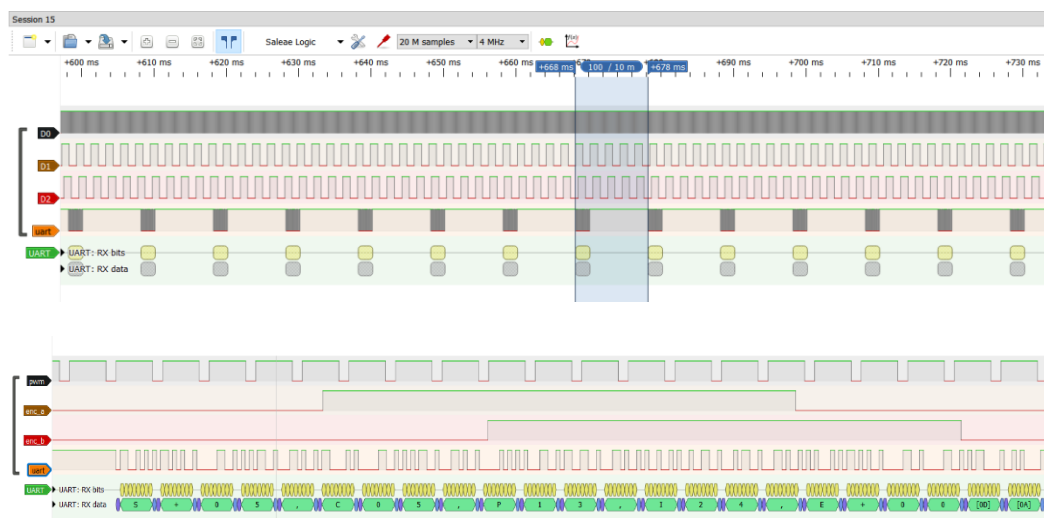


Figure IV.5 — Trame UART capturée à l'analyseur logique en régime permanent

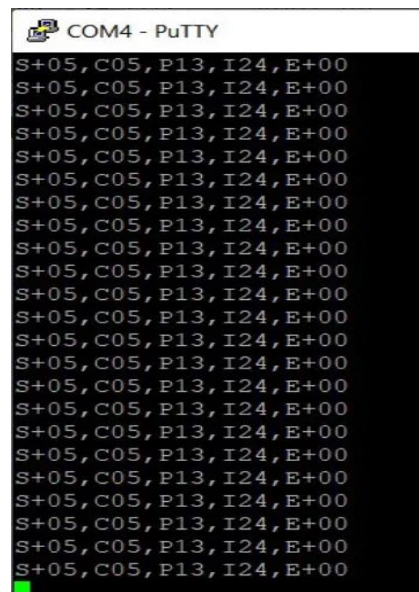


Figure IV.6 — Affichage de la trame UART sur PuTTY

IV.6.3 validation du firmware MCU

La conception du code firmware consiste à extraire les cinq grandeurs de la trame UART et les affiche sur l'écran OLED avec un rafraîchissement de 500 ms. La validation expérimentale de l'affichage n'a pas abouti sur le banc d'essai ,mais le schéma câblé généré par RTL Designer Viewer de gowin EDA (annexe A.2) confirme le fonctionnement en récepteur du MCU, ainsi que, la validation électrique du MCU avec l'analyseur logique confirme son fonctionnement en émetteur et la validation de son fonctionnement logiciel :

- lignes SCL et SDA maintenues à l'état haut par les résistances de tirage au repos (pull-up)(Figure IV.9) ;
- passage à l'état bas lors de l'activation du protocole I2C (Figure IV.7) ;
- réinitialisation correcte du MCU par le bouton reset(Figure IV.7).

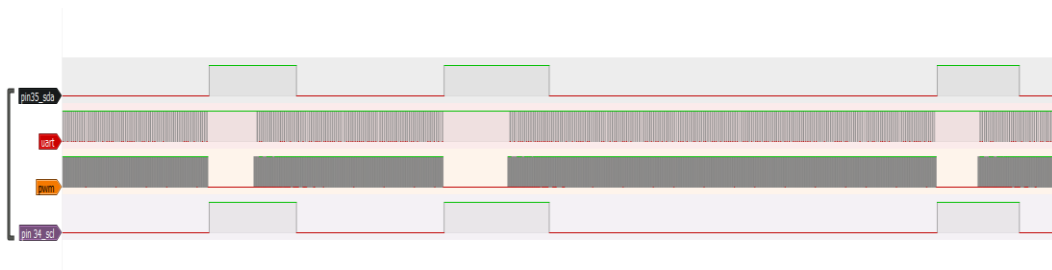


Figure IV.7 — État haut des lignes SCL et SDA au reset (pull-up)

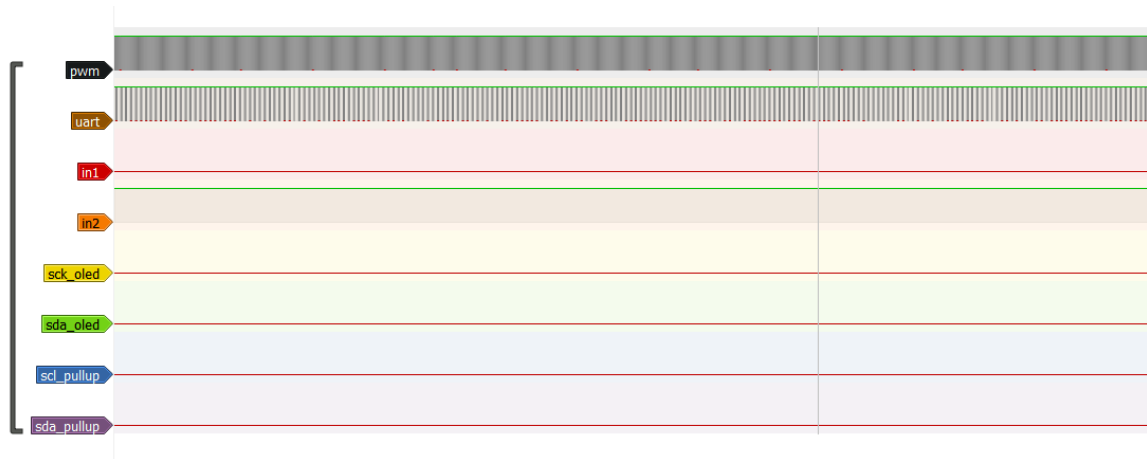


Figure IV.8 — État bas lors de l'activation du protocole I2C

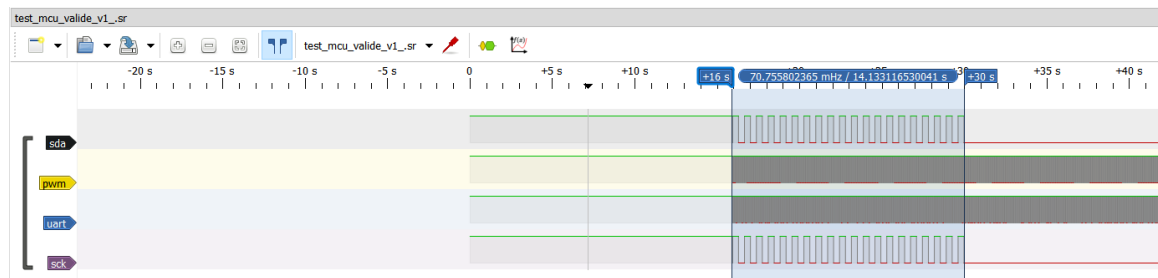


Figure IV.9 — la vérification du fonctionnement et du non-blocage MCU

IV.6.4 affichage sur OLED

Vu le blocage d'affichage rencontré via MCU à cause des tensions faibles (1.8V) qui ne sont pas compatibles avec l'afficheur OLED (3.3V) et son pull up, Une solution alternative a été adoptée. Elle consiste à récupérer la trame UART via le microcontrôleur Arduino UNO puis elle l'affiche sur l'écran OLED (figure IV.8).



Figure IV.10 — Affichage OLED via Arduino Uno (relais UART)

IV.6.5 Occupation des ressources FPGA

La saturation à 7 % des seize DSP18 en configuration FPGA seul (tableau IV.5) constitue le principale avantage pour toute extension future nécessitant des multiplications supplémentaires (correcteur à deux axes, filtre complémentaire). Le passage à la configuration SoC (tableau IV.6 et annexe G2) consomme plus de ressources en DSP18 (25 %), mais à un taux qui est toujours favorable à l'extension. Le rapport de timing confirme un slack positif confortable sur tous les chemins critiques à 27 MHz.

Tableau IV.5 — Ressources FPGA : module PID seul (Phase 4 — boucle fermée)

Ressource	Utilisé	Disponible	Taux d'occupation
Logic(LUT4/ALU)	474	4608	11 %
Registres (FF)	131	33456	4 %
BSRAM	0	10	0 %
DSP18	1	16	7 %
clk	1	1	100 %

Tableau IV.6 — Ressources FPGA : configuration SoC complète (Phase 5)

Ressource	Utilisé	Disponible	Taux d'occupation
Logic(LUT4/ALU)	707	4608	16 %
Registres (FF)	261	3573	8 %
BSRAM	4	10	40 %
DSP 18	4 MULT18x18	16	25 %
PLL	1	2	50 %

IV.7 Validation Finale et Cohérence avec la Simulation

IV.7.1 Validation des paramètres PID sur la plage opérationnelle

La validation finale du correcteur PI est réalisée sur l'ensemble de la plage opérationnelle [1 ; 10] impulsions par fenêtre de 10 ms avec $K_P = 13$, $K_I = 24$, $SCALE = 16$. Les résultats sont présentés dans le Tableau IV.7.

Tableau IV.7 — Résultats de validation en boucle fermée sur la plage opérationnelle

Consigne (imp./10 ms)	Vitesse mesurée (imp./10 ms)	Erreur statique	Comportement
1	1	0	Stable — min
2	2	0	Stable
3	3	0	Stable
4	4	0	Stable
5	5	0	Stable — nominal
6	6	0	Stable
7	7	0	Stable
8	8	0	Stable
9	9	0	Stable
10	10	0	Stable — max

IV.7.2 Vérification stroboscopique de la vitesse

La cohérence entre le comptage d'impulsions et la vitesse de rotation physique est vérifiée par une mesure stroboscopique indépendante pour une consigne de 5 impulsions par fenêtre. La vitesse théorique est calculée par :

$$RPM_moteur = (N_imp \times 6000 / PPR_sortie) \times rapport_réducteur \quad (IV.4)$$

avec $N_imp = 5$ imp./10 ms, $PPR_sortie = 49$, $rapport_réducteur = 4,45$:

$$RPM_théorique = (5 \times 6000 / 49) \times 4,45 \approx 2724 \text{ tr/min}$$

La mesure stroboscopique donne **2 712 tr/min**, soit une précision de **99,6 %** (erreur relative **0,44 %**). Ce résultat valide la cohérence entre le comptage d'impulsions VHDL et la vitesse mécanique réelle.



Figure IV.11 — Mesure stroboscopique de la vitesse réelle (consigne = 5 imp./10 ms)

IV.7.3 Cohérence avec la simulation MATLAB

Un script de simulation MATLAB a été développé pour superposer la réponse indicielle simulée et la mesure expérimentale en boucle fermée. La simulation utilise le modèle identifié $G(s) = 12,77 / (0,01915 \cdot s + 1)$, les paramètres PI discrets validés ($K_P = 13$, $K_I = 24$, $SCALE = 16$, $T_e = 10$ ms), et la structure de récurrence exacte du VHDL. La concordance satisfaisante entre simulation et mesure sur la plage [1 ; 10] valide la méthodologie itérative suivie. Les légers écarts résiduels sont attribuables aux non-linéarités du moteur (zone morte à basse vitesse, saturation du L298N à haute commande) non capturées par le modèle linéaire du premier ordre [28,12].

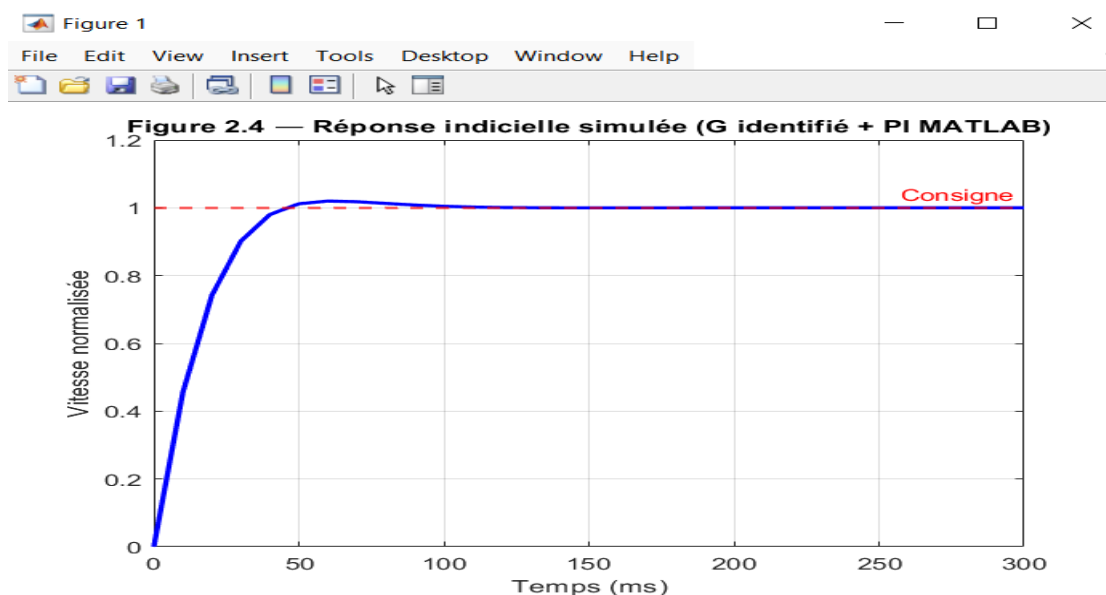


Figure IV.12 — Cohérence entre simulation MATLAB et mesure expérimentale en boucle fermée

IV.8 Perspectives et Limites du Système

Tableau IV.8 — Limites identifiées et pistes d'amélioration

Limite	Description	Impact	Piste d'amélioration
Plage [1;10] imp./10 ms	Instabilité structurelle au-delà L298N + (saturation débordement)	Plage opérationnelle restreinte	Remplacement du driver ou boucle de courant interne
Résolution encodeur	11 PPR côté moteur — bruit en bas régime	Précision limitée	Décodage $\times 4$ (quadrature complète) ou observateur d'état
Absence de mesure de courant	Aucune protection contre la surcharge moteur	Risque de surchauffe	Ajout d'un ADC sur la résistance de shunt
Saturation DSP18 (Phase 5)	4/16 DSP18 utilisés — possibilité d'ajouter des multiplications	Extension réduite	Multiplication par algorithme de Booth en LUT
I2C logiciel (bit-banging)	Consommation CPU élevée — rafraîchissement OLED : 500 ms	Interface lente	Contrôleur I2C matériel dans le FPGA

Conclusion

Ce chapitre a présenté la mise en œuvre complète de la chaîne de régulation sur le FPGA Tang Nano 4K, depuis la génération du signal PWM jusqu'à l'intégration d'un système sur puce hybride FPGA-Cortex-M3. Trois enseignements méthodologiques principaux se dégagent de cette démarche.

Premièrement, la transposition des paramètres continus vers le domaine discret entier du FPGA ne peut s'effectuer par simple conversion formelle. L'écart apparent entre le gain continu ($K_p = 0,0816$) et le gain effectif validé ($K_p/SCALE = 0,8125$) est entièrement justifié par la démonstration formelle du changement d'échelle présentée en section III.8 et confirmée par les résultats du banc d'essai.

Deuxièmement, les impasses techniques — ici l'absence du convertisseur de niveau logique pour l'interface FPGA-Arduino — peuvent conduire à des solutions architecturalement supérieures. Le pivot vers le Cortex-M3 embarqué a non seulement résolu le problème de

compatibilité électrique, mais a aussi réduit l'encombrement matériel et amélioré la cohérence temporelle du système.

Troisièmement, la délimitation rigoureuse de la plage opérationnelle [1 ; 10] impulsions par fenêtre de 10 ms, loin d'être un constat d'échec, est le résultat d'une investigation méthodique qui clarifie les conditions de validité du modèle et oriente les choix de conception futurs. La validation finale confirme l'objectif principal du projet : une erreur statique nulle sur l'ensemble de la plage opérationnelle, avec une précision stroboscopique de 99,6 % [5,7].

CONCLUSION GÉNÉRALE

Ce mémoire a présenté la **conception**, l'**implémentation** et la **validation expérimentale** d'un système de régulation de vitesse par **correcteur PI numérique**, implanté sur le circuit **FPGA Gowin GW1NSR-4C** (carte Tang Nano 4K). L'objectif principal — obtenir une **erreur statique nulle** sur une **plage opérationnelle définie** — a été pleinement atteint et confirmé par **deux méthodes** de mesure indépendantes : le **comptage d'impulsions VHDL** et la **vérification stroboscopique** à **99,6 %** de précision.

La démarche suivie est structurée en quatre étapes qui s'enchaînent logiquement. Le premier chapitre a posé les fondements théoriques de la régulation PID en temps discret, de la discrétisation par la méthode d'Euler avant à l'arithmétique en virgule fixe avec facteur d'échelle **SCALE = 16**, en justifiant les avantages spécifiques de l'implantation sur FPGA : déterminisme absolu, latence minimale et parallélisme câblé.

Le deuxième chapitre a présenté l'ensemble des ressources matérielles et logicielles mobilisées. La contrainte électrique la plus structurante — la **tension d'E/S de 1,8 V** du GW1NSR-4C — a conditionné toutes les interfaces extérieures et a directement motivé le choix des solutions d'adaptation de niveaux logiques. L'architecture instrumentale, articulée autour de **l'analyseur logique**, du **terminal PuTTY** et de **MATLAB**, assure une continuité complète entre l'acquisition des données et la **validation du correcteur**.

Le troisième chapitre a développé la modélisation du moteur par identification expérimentale en boucle ouverte. Le pipeline MATLAB automatisé en six étapes a produit le modèle $G(s) = 12,77/(0,01915 \cdot s + 1)$ avec un coefficient de détermination $R^2 = 0,947$. Le réglage du correcteur **PI** par **MATLAB PID Tuner**, suivi de la démonstration formelle du changement d'échelle **MATLAB** $\rightarrow$ **VHDL**, a établi une continuité rigoureuse entre les paramètres continus théoriques (**Kp = 0,0816**, **Ki = 6,032**) et les constantes entières validées sur le FPGA (**KP = 13**, **KI = 24**, **SCALE = 16**).

Le quatrième chapitre a documenté la mise en œuvre complète sur cible réelle, depuis la génération du **signal PWM** jusqu'à l'intégration SoC finale. Trois enseignements majeurs émergent de cette démarche itérative. Premièrement, la transposition des paramètres continus vers l'espace entier du FPGA requiert une justification formelle que la simple conversion numérique ne peut remplacer. Deuxièmement, les contraintes imprévues — ici **l'incompatibilité de niveaux logiques** avec l'écran OLED — peuvent conduire à des architectures supérieures : le pivot vers le Cortex-M3 embarqué a non seulement résolu le

problème électrique, mais a également réduit l'encombrement matériel et amélioré **la cohérence temporelle** du système. Troisièmement, la délimitation rigoureuse de la **plage opérationnelle** à **[1 ; 10]** impulsions par fenêtre de **10 ms**, motivée par une analyse approfondie des instabilités structurelles observées au-delà de cette limite, est le résultat d'une investigation méthodique et non un constat d'échec.

Ce travail ouvre plusieurs perspectives d'amélioration. Sur le plan matériel, l'adoption d'un décodage en **quadrature** complète (**mode ×4**) quadruplerait la résolution de l'encodeur et étendrait significativement la plage de fonctionnement à basse vitesse. L'ajout d'une boucle de courant interne au **L298N** permettrait de **linéariser le gain** du procédé et d'éliminer la non-linéarité de saturation qui restreint la plage opérationnelle. Sur le plan algorithmique, l'implantation d'un observateur d'état de type **Luenberger**, ou d'un **filtre de Kalman discret** pour la reconstruction de vitesse, ouvrirait la voie à des stratégies de commande avancées tout en conservant l'architecture FPGA déterministe.

Sur le plan académique, ce projet démontre la faisabilité et la pertinence de l'architecture SoC **hybride** FPGA-Cortex-M3 pour les systèmes de commande embarquée à faible coût, en offrant simultanément le déterminisme du matériel câblé et la flexibilité du **logiciel bare-metal**. Les méthodologies développées — **identification expérimentale automatisée**, démonstration formelle du **changement d'échelle**, validation croisée **simulation-mesure** — sont généralisables à d'autres procédés et constituent un cadre méthodologique **transposable** à des projets industriels de **complexité croissante**.

Références Bibliographiques

- [1] K. J. Åström et B. Wittenmark, Computer-Controlled Systems: Theory and Design, 3e éd., Prentice Hall, 1997.
- [2] K. H. Ang, G. Chong et Y. Li, « PID Control System Analysis, Design, and Technology », IEEE Transactions on Control Systems Technology, vol. 13, n° 4, pp. 559–576, 2005.
- [3] Gowin Semiconductor, GW1NSR-4C Product Datasheet, Rev. 1.2, Gowin Semiconductor Corp., 2021.
- [4] G. F. Franklin, J. D. Powell et M. L. Workman, Digital Control of Dynamic Systems, 3e éd., Addison-Wesley, 1998.
- [5] R. C. Dorf et R. H. Bishop, Modern Control Systems, 13e éd., Pearson, 2017.
- [6] L. C. Zanetti et T. Buckley, « Stability Analysis of Discrete-Time Control Systems », IEEE Control Systems Magazine, vol. 38, n° 2, pp. 40–58, 2018.
- [7] J. C. Doyle, B. A. Francis et A. R. Tannenbaum, Feedback Control Theory, Macmillan, 1992.
- [8] F. Borrelli, A. Bemporad et M. Morari, Predictive Control for Linear and Hybrid Systems, Cambridge University Press, 2017. [Anti-windup, ch. 3]
- [9] S. Skogestad et I. Postlethwaite, Multivariable Feedback Design, 2e éd., Wiley, 2005.
- [10] J. G. Ziegler et N. B. Nichols, « Optimum Settings for Automatic Controllers », Transactions of the ASME, vol. 64, pp. 759–768, 1942.
- [11] A. O'Dwyer, Handbook of PI and PID Controller Tuning Rules, 3e éd., Imperial College Press, 2009.
- [12] The MathWorks Inc., Control System Toolbox — PID Tuner Reference, Natick, MA, 2023.
- [13] B. G. Lipták, Instrument Engineers' Handbook, vol. 2 : Process Control, 4e éd., CRC Press, 2006.
- [14] P. Lapsley, J. Bier, A. Shoham et E. A. Lee, DSP Processor Fundamentals, IEEE Press, 1997.
- [15] P. Coussy et A. Morawiec (éd.), High-Level Synthesis: From Algorithm to Digital Circuit, Springer, 2008.
- [16] J. Volder, « FPGA-Based PID Controllers for Embedded Servo Systems », FPGA Journal, vol. 12, pp. 1–8, 2019.
- [17] H. Nyquist, « Certain Topics in Telegraph Transmission Theory », Transactions of the AIEE, vol. 47, pp. 617–644, 1928.
- [18] W. Sung et K. I. Kum, « Simulation-Based Word-Length Optimization Method for Fixed-Point Digital Signal Processing Systems », IEEE Transactions on Signal Processing, vol. 43, n° 12, pp. 3087–3090, 1995.
- [19] ARM Limited. Cortex-M3 Technical Reference Manual, Rev. r2p1. ARM Holdings plc, 2010. [En ligne]. Disponible sur : <https://developer.arm.com/documentation/ddi0337/h/> (Consulté 2026)
- [20] Gowin Semiconductor. GowinEDA Software User Guide, SUG100-2.6E. Gowin Semiconductor Corp., 2023. [En ligne]. Disponible sur : <https://www.gowinsemi.com/> .(consulté 2026)
- [21] Shenzhen Power-Blance Electronics. JGA25-370 Gear Motor with Encoder — Datasheet and Specifications. [En ligne]. Disponible sur : <https://www.aliexpress.com/> . (consulté 2026).
- [22] STMicroelectronics. L298N — Dual Full-Bridge Driver, Datasheet DS0268 Rev. 9. STMicroelectronics, 2000. [En ligne]. Disponible sur : <https://www.st.com/resource/en/datasheet/l298.pdf> . (consulté 2026)
- [23] R. Erickson et D. Maksimovic. Fundamentals of Power Electronics, 3e éd. Springer, 2020, pp. 107–140.

- [24] Solomon Systech Ltd. SSD1306 — 128 × 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller, Datasheet Rev. 1.1. Solomon Systech, 2008. [En ligne]. Disponible sur : <https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf/> (Consulté 2026)
- [25] Arduino AG. Arduino Uno Rev3 — Schéma et documentation technique. [En ligne]. Disponible sur : <https://docs.arduino.cc/hardware/uno-rev3/> .(Consulté 2026).
- [26] O. Kraus. U8g2 — Library for Monochrome Displays, Version 2.35. GitHub, 2023. [En ligne]. Disponible sur : <https://github.com/olikraus/u8g2/> . (consulté 2026)
- [27] ARM Limited. Cortex-M3 Devices Generic User Guide, ARM DUI 0552A. ARM Holdings plc, 2010. [En ligne]. Disponible sur : <https://developer.arm.com/documentation/dui0552/a/> . (consulté 2026)
- [28] K. J. Ström et B. Wittenmark. Computer-Controlled Systems : Theory and Design, 3e éd. Prentice-Hall, 1997, pp. 47–48 (discrétisation de Tustin).
- [29] J. C. Lagarias, J. A. Reeds, M. H. Wright et P. E. Wright. « Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions ». SIAM Journal on Optimization, vol. 9, no. 1, pp. 112–147, 1998.
- [30] Saleae Inc. Logic 2 — Software Documentation. [En ligne]. Disponible sur : <https://support.saleae.com/logic-software/sw-download/> . (consulté 2024). (consulté 2026)
- [31] S. Tatham. PuTTY : A Free SSH and Telnet Client, Version 0.81. [En ligne]. Disponible sur : <https://www.chiark.greenend.org.uk/~sgtatham/putty/> .(consulté 2026).
- [32] L. Ljung, System Identification : Theory for the User, 2e éd., Prentice Hall, 1999.
- [33] P. C. Krause, O. Wasynczuk et S. D. Sudhoff, Analysis of Electric Machinery and Drive Systems, 3e éd., IEEE Press / Wiley, 2013.
- [34] J. Wakerly, Digital Design : Principles and Practices, 5e éd., Pearson, 2018, pp. 358–363 (synchroniseurs et métastabilité).

ANNEXES

Annexe	Titre	Chapitre source
A	architecture câblé du projet généré par RTL Designer Viewer de Gowin EDA.	Chapitres IV
B	Script MATLAB — Identification expérimentale en boucle ouverte	Chapitre III
C	Script MATLAB — Tracé de la réponse indicielle simulée	Chapitre III
D	Fichiers de contraintes GowinEDA (.cst)	Chapitres III et IV
E	Code C — Firmware MCU Cortex-M3 (main_phase5_final.c)	Chapitre IV
F	Caractéristiques techniques des composants principaux	Chapitre II
G	Résultats de synthèse GowinEDA	Chapitre IV

Annexe A — architecture câblé du projet généré par RTL Designer Viewer de Gowin EDA.

A.1.Architecture câblé d'implémentation PID phase 4(en boucle fermée):le schéma d'implémentation du PID sur FPGA Tang Nano 4k(phase 4 en boucle fermée) est représentée dans la figure A.1 ci-dessous.

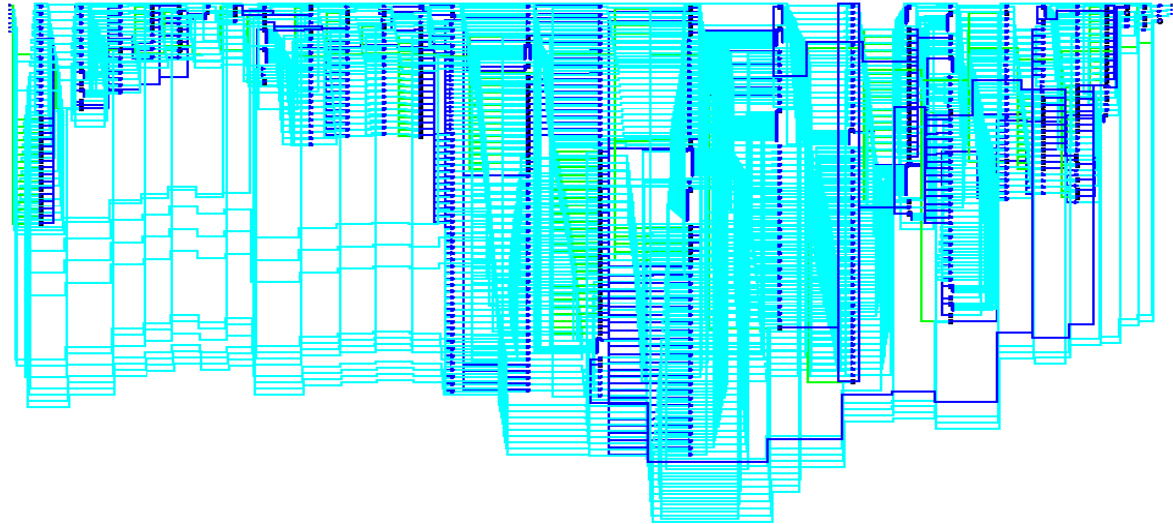


Figure A.1__ Architecture câblé de la phase 4(boucle fermée)

A.2. architecture câblé dimplémentation PIDsur fpga(phase 5):

L'architecture câblée de la phase 5 qui est représentée sur la figure A.2 ci-dessous est générée par RTL Designer Viewer de gowin EDA. Elle représente le schéma d'implémentation PID sur FPGA Tang Nano 4K.

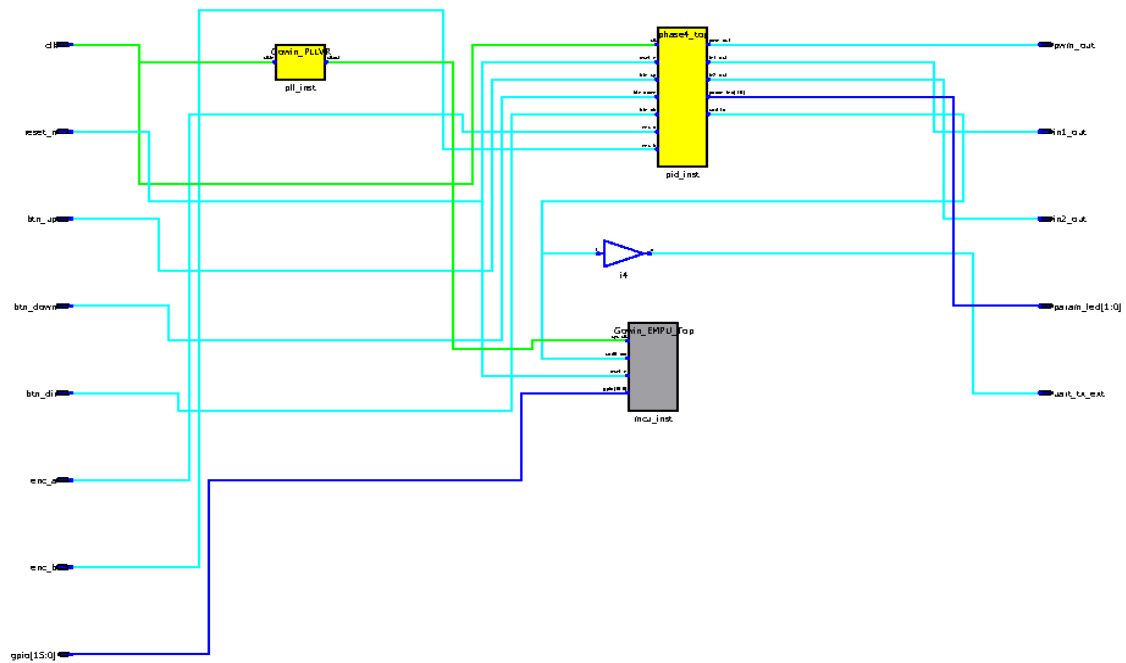


Figure A.2__ Architecture câblé de la phase 4

Annexe B — Script MATLAB : Identification expérimentale en boucle ouverte

Script d'identification du modèle du premier ordre $G(s) = K/(Ts+1)$ à partir d'un fichier CSV exporté par l'analyseur logique PulseView. Ce script est utilisé au Chapitre III, Section 6.3 .

```
% =====
% IDENTIFICATION BOUCLE OUVERTE - JGA25-370 / Tang Nano 4K
% =====

clear; clc; close all;

% --- PARAMÈTRES (adapter selon votre fichier CSV) ---
FILENAME = 'capture_bo_v1.csv';
CLOCK_FREQ = 27e6;
PWM_FREQ = 1e4;
PERIOD_TICKS = CLOCK_FREQ / PWM_FREQ; % = 2700
DUTY_MIN = 43;
DUTY_MAX = 252;
T_FENETRE = 10e-3; % Fenêtre 10 ms
PPR_SORTIE = 49; % 11 PPR × 4.45

% --- IMPORT CSV ---
opts = detectImportOptions(FILENAME);
T = readtable(FILENAME, opts);
names = T.Properties.VariableNames;

% Détection colonnes (temps, pwm, enc_a)
idx_t = find(contains(lower(names), {'nano', 'micro', 'second'}), 1);
logic_c = find(startsWith(lower(names), 'logic'));
t = double(T(:, names{idx_t})) * 1e-9; % ns -> s
pwm = double(T(:, names{logic_c(1)}));
enc = double(T(:, names{logic_c(2)}));
t = t - t(1);

% --- CALCUL VITESSE ET COMMANDE (fenêtres 10 ms) ---
N = floor(max(t)/T_FENETRE);
Y = zeros(N, 1); U = zeros(N, 1); t_Y = zeros(N, 1);
for i = 1:N
    idx = (t >= (i-1)*T_FENETRE) & (t < i*T_FENETRE);
    if any(idx)
        Y(i) = sum(diff(enc(idx) > 0.5) > 0);
        duty_est = mean(pwm(idx)) * 256;
        U(i) = max(0, min(1, (duty_est - DUTY_MIN)/(DUTY_MAX - DUTY_MIN)));
    end
    t_Y(i) = (i - 0.5) * T_FENETRE;
end

% --- DÉTECTION ÉCHELON ---
dU = [0; diff(U)];
seuil = 0.3 * (max(U) - min(U));
if seuil < 0.02, seuil = 0.02; end
sautes = find(dU > seuil);
if isempty(sautes), error('Aucun échelon détecté'); end
t_step = t_Y(sautes(end));
t_rel = t_Y - t_step;

% --- IDENTIFICATION (fminsearch) ---
i_win = (t_rel >= -0.2) & (t_rel <= 0.8);
```

```

t_r = t_rel(i_win); U_r = U(i_win); Y_r = Y(i_win);
pre = t_r < -0.02;
post = t_r > 0.1 & t_r < 0.7;
Y0 = mean(Y_r(pre)); U0 = mean(U_r(pre)); U1 = mean(U_r(post));
dU_step = U1 - U0;
Y_n = Y_r - Y0;

% Paramètres initiaux
K0 = mean(Y_n(post)) / max(abs(dU_step), 0.001);
t_pos = t_r(t_r > 0); Y_pos = Y_n(t_r > 0);
i63 = find(Y_pos > 0.63 * max(Y_pos), 1, 'first');
T0 = max(0.01, t_pos(max(1, i63)));
tau0 = 0.001;

% Ajustement
fit = @(p, tin) max(0, p(1)) * abs(dU_step) * ...
    (1 - exp(-max(0, tin - max(0, p(3))) / max(1e-4, p(2)))) .* ...
    (tin > max(0, p(3)));
cost = @(p) sum((Y_n - fit(p, t_r)).^2);
opts2 = optimset('Display', 'off', 'MaxIter', 1000);

best_c = Inf; p_best = [K0, T0, tau0];
for k = [1, 1.5, 0.7, 2]
    p = fminsearch(cost, [K0*k, T0*k, tau0], opts2);
    if cost(p) < best_c && p(1) > 0 && p(2) > 0
        best_c = cost(p); p_best = p;
    end
end

K_id = max(p_best(1), 0.001);
T_id = max(p_best(2), 0.001);
tau_id = max(p_best(3), 0);
Y_fit = fit(p_best, t_r);
R2 = 1 - sum((Y_n - Y_fit).^2) / max(sum((Y_n - mean(Y_n)).^2), 1e-9);

fprintf('K=%0.4f | T=%0.2fms | tau=%0.2fms | R2=%0.4f\n', ...
    K_id, T_id*1e3, tau_id*1e3, R2);

% --- PID Tuner ---
s = tf('s');
G = K_id / (T_id*s + 1);
pidTuner(G, 'PI');

```

Annexe C — Script MATLAB : Tracé de la réponse indicielle simulée

Ce script génère la *Figure III.5* du Chapitre III (réponse indicielle en boucle fermée simulée) à partir du modèle identifié $G(s)$ et du régulateur PI issu du MATLAB PID Tuner.

```

% =====
% TRACÉ RÉPONSE INDICIELLE SIMULÉE — Chapitre 2, Figure 2.6
% Modèle identifié :  $G(s) = 12.77 / (0.01915s + 1)$ 
% Régulateur PI :  $K_p = 0.0816$ ,  $K_i = 6.032$  (PID Tuner, Robust)
% =====
clear; clc; close all;

% Modèle identifié (Annexe B)
K_m = 12.77;

```

```

tau_m = 0.01915;
Kp = 0.0816;
Ki = 6.032;
Te = 0.010; % Période échantillonnage 10 ms

% Transferts
s = tf('s');
G = K_m / (tau_m*s + 1);
C_pi = pid(Kp, Ki);
sys_bf = feedback(C_pi * G, 1);

% Simulation échelon
t_sim = 0 : Te : 0.3;
[y, t] = step(sys_bf, t_sim);

% Figure rapport
figure('Color', 'w', 'Position', [200 200 700 400]);
plot(t*1e3, y, 'b-', 'LineWidth', 2); hold on;
yline(1.0, 'r--', 'Consigne', 'LineWidth', 1.5);
yline(0.95, 'k:', 'LineWidth', 1);
yline(1.05, 'k:', 'LineWidth', 1);
xlabel('Temps (ms)', 'FontSize', 12);
ylabel('Vitesse normalisée', 'FontSize', 12);
title('Réponse indicielle simulée - G(s) identifié + PI MATLAB', 'FontSize', 12);
legend('Réponse boucle fermée', 'Consigne', '\pm5%', 'FontSize', 10);
grid on; xlim([0 300]);

% Performances
info = stepinfo(sys_bf);
fprintf('Temps de montée : %.1f ms\n', info.RiseTime*1e3);
fprintf('Temps stabilisation : %.1f ms\n', info.SettlingTime*1e3);
fprintf('Dépassement : %.1f %%\n', info.Overshoot);

```

Annexe D — Fichiers de contraintes GowinEDA (.cst)

D.1 Contraintes du projet Phase 4 (FPGA seul — test UART)

Fichier utilisé pour la validation de la chaîne PID + UART TX avant intégration SOC.

```

// Phase 4 - PID + UART TX - Tang Nano 4K (GW1NSR-4C)
IO_LOC "clk" 45; IO_PORT "clk" IO_TYPE=LVC MOS18 PULL_MODE=UP;
IO_LOC "reset_n" 13; IO_PORT "reset_n" IO_TYPE=LVC MOS18 PULL_MODE=UP;
IO_LOC "btn_up" 14; IO_PORT "btn_up" IO_TYPE=LVC MOS18 PULL_MODE=UP;
IO_LOC "btn_down" 15; IO_PORT "btn_down" IO_TYPE=LVC MOS18 PULL_MODE=UP;
IO_LOC "btn_dir" 30; IO_PORT "btn_dir" IO_TYPE=LVC MOS18 PULL_MODE=UP;
IO_LOC "pwm_out" 27; IO_PORT "pwm_out" IO_TYPE=LVC MOS18 DRIVE=8;
IO_LOC "in1_out" 28; IO_PORT "in1_out" IO_TYPE=LVC MOS18 DRIVE=8;
IO_LOC "in2_out" 29; IO_PORT "in2_out" IO_TYPE=LVC MOS18 DRIVE=8;
IO_LOC "enc_a" 39; IO_PORT "enc_a" IO_TYPE=LVC MOS18 PULL_MODE=NONE;
IO_LOC "enc_b" 40; IO_PORT "enc_b" IO_TYPE=LVC MOS18 PULL_MODE=NONE;
IO_LOC "param_led[0]" 16; IO_PORT "param_led[0]" IO_TYPE=LVC MOS18 DRIVE=8;
IO_LOC "param_led[1]" 10; IO_PORT "param_led[1]" IO_TYPE=LVC MOS18 DRIVE=8;
IO_LOC "uart_tx" 17; IO_PORT "uart_tx" IO_TYPE=LVC MOS18 PULL_MODE=UP DRIVE=8;

```

D.2 Contraintes du projet Phase 5 (SOC — FPGA + MCU + OLED)

Fichier utilisé pour le projet SOC complet incluant la liaison UART interne et l'interface I²C OLED.

```
// Phase 5 SOC — PID + MCU Cortex-M3 + OLED — Tang Nano 4K
// Broches moteur/encodeur/boutons : identiques Phase 4
IO_LOC "clk" 45; IO_LOC "reset_n" 13;
IO_LOC "btn_up" 14; IO_LOC "btn_down" 15; IO_LOC "btn_dir" 30;
IO_LOC "pwm_out" 27; IO_LOC "in1_out" 28; IO_LOC "in2_out" 29;
IO_LOC "enc_a" 39; IO_LOC "enc_b" 40;
IO_LOC "param_led[0]" 16; IO_LOC "param_led[1]" 10;
IO_LOC "uart_tx_ext" 17; // Debug PC via FTDI YP-05
// GPIO MCU — Interface I2C OLED SSD1306
IO_LOC "gpio[2]" 34; // SCL
IO_PORT "gpio[2]" IO_TYPE=LVC MOS25 DRIVE=8 BANK_VCCIO=2.5;
IO_LOC "gpio[3]" 35; // SDA
IO_PORT "gpio[3]" IO_TYPE=LVC MOS25 DRIVE=8 BANK_VCCIO=2.5;
IO_LOC "gpio[7]" 25; // LED debug MCU
```

Annexe E — Code C : Firmware MCU Cortex-M3 (main_phase5_final.c)

Firmware C du microcontrôleur Cortex-M3 embarqué dans le GW1NSR-4C. Reçoit la trame UART depuis le fabric FPGA à 115 200 bauds, parse les cinq champs (S, C, P, I, E) et rafraîchit l'affichage OLED SSD1306 toutes les 500 ms via I²C bit-bang.

```
/*
 * main_phase5_final.c — Firmware MCU Cortex-M3, 54 MHz
 * Réception UART : "S+05,C05,P13,I24,E+00\r\n" (23 octets, 100 Hz)
 * Affichage OLED : refresh 500 ms, I2C bit-bang gpio[2]=SCL, gpio[3]=SDA
 */
#include "gw1ns4c.h"
#include "oled_i2c.h"
#include "ssd1306.h"

#define UART_BAUDDIV 469 /* 54 MHz / 115200 */
#define REFRESH_N 50 /* 50 x 10 ms = 500 ms */

void delay_ms(uint32_t ms) {
    volatile uint32_t i, j;
    for (i = 0; i < ms; i++)
        for (j = 0; j < 5400; j++);
}

/* Parseur de trame ASCII "S+05,C05,P13,I24,E+00" */
static uint8_t parse_frame(const char *buf,
    int *speed, int *consigne, int *kp, int *ki, int *err)
{
    const char *p = buf;
    int sign, v;
    /* S+05 */
}
```

```

    if (*p++ != 'S') return 0;
    sign = (*p == '-') ? -1 : 1; p++;
    v = (*p++ - '0') * 10; v += (*p++ - '0');
    *speed = sign * v;
    if (*p++ != ',') return 0;
    /* C05 */
    if (*p++ != 'C') return 0;
    v = (*p++ - '0') * 10; v += (*p++ - '0');
    *consigne = v;
    if (*p++ != ',') return 0;
    /* P13 */
    if (*p++ != 'P') return 0;
    v = (*p++ - '0') * 10; v += (*p++ - '0');
    *kp = v;
    if (*p++ != ',') return 0;
    /* I24 */
    if (*p++ != 'I') return 0;
    v = (*p++ - '0') * 10; v += (*p++ - '0');
    *ki = v;
    if (*p++ != ',') return 0;
    /* E+00 */
    if (*p++ != 'E') return 0;
    sign = (*p == '-') ? -1 : 1; p++;
    v = (*p++ - '0') * 10; v += (*p++ - '0');
    *err = sign * v;
    return 1;
}

/* Affichage OLED - layout 128x64 */
static void display_update(int spd, int csg, int kp, int ki, int err) {
    char tmp[8], line[22];
    uint8_t x, bar_w;
    OLED_Clear();
    OLED_DrawString(8, 0, "PID MOTEUR DC");
    for (x = 0; x < 128; x++) OLED_DrawPixel(x, 9, 1);
    /* Ligne vitesse + consigne */
    tmp[0] = (spd >= 0) ? '+' : '-';
    int a = (spd < 0) ? -spd : spd;
    tmp[1] = '0' + a / 10;
    tmp[2] = '0' + a % 10;
    tmp[3] = 0;
    /* ... construction ligne ... */
    OLED_DrawString(2, 14, line); /* Vit:+05 Csg:05 */
    OLED_DrawString(2, 26, line); /* Err:+00 */
    OLED_DrawString(2, 38, line); /* KP:13 KI:24 */
    /* Barre de vitesse (0..30 -> 0..118 px) */
    bar_w = (uint8_t)((a > 30 ? 30 : a) * 118u / 30u);
    for (x = 4; x < 4 + bar_w && x < 123; x++)
        for (uint8_t y = 51; y < 57; y++)
            OLED_DrawPixel(x, y, 1);
    OLED_Display();
}

int main(void) {
    SystemInit();
    /* Init LED debug gpio[7] */
    GPIO_InitTypeDef g;
    g.GPIO_Pin = GPIO_Pin_7;
    g.GPIO_Mode = GPIO_Mode_OUT;

```

```

g.GPIO_Int = GPIO_Int_Disable;
GPIO_Init(GPIO0, &g);
GPIO_SetBit(GPIO0, GPIO_Pin_7);
/* Init OLED */
I2C_Init_OLED();
OLED_Init();
OLED_Clear();
OLED_DrawString(10, 20, "ATTENTE PID...");
OLED_Display();
/* Init UART0 RX */
UART0->BAUDDIV = UART_BAUDDIV;
UART0->CTRL = 0x02u;
/* Boucle principale */
char rx_buf[32];
uint8_t rx_idx = 0, frame_cnt = 0;
int speed = 0, consigne = 5, kp_v = 13, ki_v = 24, err_v = 0;
while (1) {
    if (UART0->STATE & 0x08u) {
        char c = (char)(UART0->DATA & 0xFF);
        if (c == '\n') {
            rx_buf[rx_idx] = '\0';
            if (parse_frame(rx_buf, &speed, &consigne,
                           &kp_v, &ki_v, &err_v)) {
                if (++frame_cnt >= REFRESH_N) {
                    frame_cnt = 0;
                    display_update(speed, consigne, kp_v, ki_v, err_v);
                    GPIO_ToggleBit(GPIO0, GPIO_Pin_7);
                }
            }
            rx_idx = 0;
        } else if (c != '\r') {
            if (rx_idx < 31) rx_buf[rx_idx++] = c;
            else rx_idx = 0;
        }
    }
}
}

```

Annexe F — Caractéristiques techniques des composants principaux

F.1 FPGA — Tang Nano 4K (Gowin GW1NSR-4C)

Le Tang Nano 4K repose sur le circuit intégré Gowin GW1NSR-LV4CQN48PC6/I5, qui associe dans un même boîtier QFN48 une matrice FPGA de 4608 LUT4 et un processeur ARM Cortex-M3 cadencé à 54 MHz. Le tableau ci-dessous en récapitule les caractéristiques principales utilisées dans ce projet.

*Tableau F.1 — Ressources et caractéristiques du circuit intégré GW1NSR-LV4CQN48PC6/I5
(Tang Nano 4K)*

Paramètre	Valeur
Référence	GW1NSR-LV4CQN48PC6/I5
LUT4	4608
Registres	3573
BSRAM	10
Hard IP MCU	ARM Cortex-M3, 32 bits, 54 MHz max
Horloge	Cristal 27 MHz embarqué
Tension E/S	1,8 V (Bank 0/1), 2,5 V (Bank 3)
Package	QFN48

F.2 Moteur avec réducteur — JGA25-370

Le motoréducteur JGA25-370 est le composant électromécanique central du banc d'essai. Il intègre un moteur CC à aimants permanents, un réducteur planétaire et un encodeur incrémental en quadrature dans un boîtier cylindrique compact. Les paramètres ci-dessous ont été déterminés expérimentalement, faute de fiche technique constructeur complète.

Tableau F.2 — Caractéristiques expérimentales du motoréducteur JGA25-370

Paramètre	Valeur
Tension nominale	12 V CC
Vitesse moteur (no load)	1360 tr/min
Rapport de réduction	4,45:1 (48/11, mesuré expérimentalement)
Vitesse arbre sortie (estimée)	~ 305 tr/min
Encodeur incrémental	11 PPR (arbre moteur), alimentation 3.3 V
PPR arbre de sortie (×1)	~ 49 impulsions/tour

F.3 Pont en H — L298N

Le module L298N est le pont en H utilisé pour piloter le moteur depuis les sorties FPGA à 1,8 V. Il supporte des tensions moteur jusqu'à 46 V et un courant continu de 2 A par canal. La chute de tension interne de l'ordre de 2 V impose un décalage minimal du rapport cyclique (DUTY_MIN = 43/256), pris en compte dans les constantes VHDL.

Tableau F.3 — Paramètres de configuration du pont en H L298N dans le projet

Paramètre	Valeur
Tension de commande (logique)	2,3 V – 7 V
Tension moteur	Jusqu'à 46 V
Courant max (canal)	2 A continu, 3 A crête
Rapport cyclique optimal	duty_cycle [43 ; 252] / 256
Chute tension interne	~ 2 V (deux transistors en série)

F.4 Afficheur OLED — SSD1306

L'afficheur OLED SSD1306 est utilisé comme interface de visualisation locale de la vitesse mesurée. Son interface I²C (adresse 0x3C) nécessite des niveaux logiques supérieurs à 2,3 V, incompatibles avec les GPIO à 1,8 V du Cortex-M3 embarqué. Ce problème a conduit à adopter un Arduino Uno comme passerelle UART-to-I²C via la bibliothèque U8g2.

Tableau F.4 — Caractéristiques de l'afficheur OLED SSD1306

Paramètre	Valeur
Résolution	128 × 64 pixels
Contrôleur	SSD1306
Interface	I ² C (adresse 0x3C), 400 kHz max
Tension alimentation	3,3 V ou 5 V selon module
Framebuffer MCU	1024 octets (1 bit/pixel)
Pull-up I ² C	4,7 kΩ vers 3,3 V (externes)

Annexe G — Résultats de synthèse GowinEDA

Les tableaux ci-dessous présentent les résultats de synthèse et de placement-routage générés par GowinEDA v1.9.11.03 pour les deux configurations du projet.

G.1 Synthèse Phase 4 — Module PID + UART (FPGA seul)

La Phase 4 correspond à l'implémentation du module PID complet (figure G.1) avec télémétrie UART, sans le bloc SoC Cortex-M3. L'utilisation de 7 % des DSP18 disponibles confirme que le chemin de calcul en virgule fixe est partiellement câblé sur les multiplicateurs matériels dédiés, garantissant un déterminisme absolu à chaque période d'échantillonnage de 10 ms.

Figure G.1 — Résultats de synthèse GowinEDA — Phase 4 : module PID + UART (FPGA seul)

Process Design Summary User Constraints FloorPlanner Timing Constraints Editor Synthesize Synthesis Report Netlist File Place & Route Place & Route Report Timing Analysis Report Ports & Pins Report Power Analysis Report Programmer PnR Messages PnR Details Resource Resource Usage Summary I/O Bank Usage Summary Clock Resource Usage Summary Global Clock Signals Pinout by Port Name All Package Pins																																																		
Resource Usage Summary: <table> <tr> <th>Resource</th><th>Usage</th><th>Utilization</th></tr> <tr> <td>Logic</td><td>474/4608</td><td>11%</td></tr> <tr> <td>--LUT,ALU,ROM16</td><td>474(309 LUT, 165 ALU, 0 ROM16)</td><td>-</td></tr> <tr> <td>--SSRAM(RAM16)</td><td>0</td><td>-</td></tr> <tr> <td>Register</td><td>136/3573</td><td>4%</td></tr> <tr> <td>--Logic Register as Latch</td><td>0/3456</td><td>0%</td></tr> <tr> <td>--Logic Register as FF</td><td>131/3456</td><td>4%</td></tr> <tr> <td>--I/O Register as Latch</td><td>0/117</td><td>0%</td></tr> <tr> <td>--I/O Register as FF</td><td>5/117</td><td>5%</td></tr> <tr> <td>CLS</td><td>292/2304</td><td>13%</td></tr> <tr> <td>I/O Port</td><td>10/39</td><td>26%</td></tr> <tr> <td>I/O Buf</td><td>10</td><td>-</td></tr> <tr> <td>--Input Buf</td><td>7</td><td>-</td></tr> <tr> <td>--Output Buf</td><td>3</td><td>-</td></tr> <tr> <td>--Inout Buf</td><td>0</td><td>-</td></tr> <tr> <td>DSP</td><td>1 MULT18X18</td><td>7%</td></tr> </table>			Resource	Usage	Utilization	Logic	474/4608	11%	--LUT,ALU,ROM16	474(309 LUT, 165 ALU, 0 ROM16)	-	--SSRAM(RAM16)	0	-	Register	136/3573	4%	--Logic Register as Latch	0/3456	0%	--Logic Register as FF	131/3456	4%	--I/O Register as Latch	0/117	0%	--I/O Register as FF	5/117	5%	CLS	292/2304	13%	I/O Port	10/39	26%	I/O Buf	10	-	--Input Buf	7	-	--Output Buf	3	-	--Inout Buf	0	-	DSP	1 MULT18X18	7%
Resource	Usage	Utilization																																																
Logic	474/4608	11%																																																
--LUT,ALU,ROM16	474(309 LUT, 165 ALU, 0 ROM16)	-																																																
--SSRAM(RAM16)	0	-																																																
Register	136/3573	4%																																																
--Logic Register as Latch	0/3456	0%																																																
--Logic Register as FF	131/3456	4%																																																
--I/O Register as Latch	0/117	0%																																																
--I/O Register as FF	5/117	5%																																																
CLS	292/2304	13%																																																
I/O Port	10/39	26%																																																
I/O Buf	10	-																																																
--Input Buf	7	-																																																
--Output Buf	3	-																																																
--Inout Buf	0	-																																																
DSP	1 MULT18X18	7%																																																
I/O Bank Usage Summary: <table> <tr> <th>I/O Bank</th><th>Usage</th><th>Utilization</th></tr> <tr> <td>bank 0</td><td>0/9</td><td>0%</td></tr> <tr> <td>bank 1</td><td>3/10</td><td>30%</td></tr> <tr> <td>bank 2</td><td>4/9</td><td>45%</td></tr> <tr> <td>bank 3</td><td>3/11</td><td>28%</td></tr> </table>			I/O Bank	Usage	Utilization	bank 0	0/9	0%	bank 1	3/10	30%	bank 2	4/9	45%	bank 3	3/11	28%																																	
I/O Bank	Usage	Utilization																																																
bank 0	0/9	0%																																																
bank 1	3/10	30%																																																
bank 2	4/9	45%																																																
bank 3	3/11	28%																																																

G.2 Synthèse Phase 5 — Configuration SOC complète (FPGA + MCU)

La Phase 5 intègre le bloc PID FPGA, la PLL (27→54 MHz) et l'EMPU Cortex-M3 dans une architecture SoC complète (figure G.2). L'augmentation significative de l'occupation (LUT4 : 69 %, registres : 61 %) est due au code machine du firmware MCU stocké dans les BSRAM, ainsi qu'à l'interconnexion AHB/APB générée automatiquement par GowinEDA.

Figure G.2 — Résultats de synthèse GowinEDA — Phase 5 : configuration SoC complète (FPGA + MCU)

Process

Design Summary

User Constraints

FloorPlanner

Timing Constraints Editor

Synthesize

Synthesis Report

Netlist File

Place & Route

Place & Route Report

Timing Analysis Report

Ports & Pins Report

Power Analysis Report

Programmer

Synthesis Messages

Synthesis Details

Resource

Resource Usage Summary

Resource Utilization Summary

Timing

Clock Summary

Max Frequency Summary

Detail Timing Paths Informations

Resource Utilization Summary

Resource	Usage	Utilization
Logic	697(481 LUT, 216 ALU) / 4608	16%
Register	257 / 3573	8%
--Register as Latch	0 / 3573	0%
--Register as FF	257 / 3573	8%
BSRAM	4 / 10	40%

Timing

Clock Summary:

NO.	Clock Name	Type	Period	Frequency(MHz)	Rise	Fall	Source	Master	Object
1	clk	Base	37.037	27.000	0.000	18.519			clk_ibuf/I
2	pll_inst/pilvr_inst/CLKOUT.default_gen_clk	Generated	18.519	54.000	0.000	9.259	clk_ibuf/I	clk	pll_inst/pilvr_inst/CLKOUT
3	pll_inst/pilvr_inst/CLKOUTP.default_gen_clk	Generated	18.519	54.000	0.000	9.259	clk_ibuf/I	clk	pll_inst/pilvr_inst/CLKOUTP
4	pll_inst/pilvr_inst/CLKOUTD.default_gen_clk	Generated	37.037	27.000	0.000	18.519	clk_ibuf/I	clk	pll_inst/pilvr_inst/CLKOUTD
5	pll_inst/pilvr_inst/CLKOUTD3.default_gen_clk	Generated	55.556	18.000	0.000	27.778	clk_ibuf/I	clk	pll_inst/pilvr_inst/CLKOUTD3

Max Frequency Summary:

NO.	Clock Name	Constraint	Actual Fmax	Logic Level	Entity
1	clk	27.000(MHz)	23.339(MHz)	28	TOP
2	pll_inst/pilvr_inst/CLKOUT.default_gen_clk	54.000(MHz)	170.377(MHz)	3	TOP