LISV Laboratoire d'ingénierie des systèmes de Versailles | UVSQ université PARIS-SACLAY

Architecture des systèmes numériques et informatiques
PROJET NOTÉ : Calculatrice 4 bits avec UAL

Halim Djerroud

5 novembre 2025

Informations pratiques

— **Durée** : 2h30
— **Documents autorisés** : Notes personnelles du TP du matin, documentation Logisim
— **Travail** : Individuel
— **Notation** : /20 points
— **Rendu** :
  — Fichier Logisim (.circ) nommé : `NOM_Prenom_Calculatrice.circ`
  — Document PDF avec les tables de vérité, schémas et explications
  — Captures d'écran des tests réussis

IMPORTANT : Sauvegardez régulièrement votre travail ! Testez chaque composant avant de passer au suivant. La démarche et la méthodologie comptent autant que le résultat final.

Contexte du projet

Vous êtes ingénieur chez MicroCalc Inc. et devez concevoir une calculatrice 4 bits complète intégrant :
— Une UAL (Unité Arithmétique et Logique) capable d'effectuer 8 opérations
— Un système de registres pour stocker les opérandes
— Un décodeur d'affichage 7 segments
— Un système de détection d'erreurs (débordement, division par zéro)

Ce projet met en œuvre toutes les compétences acquises ce matin : circuits combinatoires, simplification par Karnaugh, additionneurs, UAL et conception modulaire.

Partie 1 : UAL 4 bits (8 points)

1.1 Spécifications de l'UAL (1 point)

Votre UAL doit implémenter les 8 opérations suivantes :

OP[2 :0]	Mnémonique	Opération	Description
000	AND	$S = A \wedge B$	ET logique
001	OR	$S = A \vee B$	OU logique
010	ADD	$S = A + B$	Addition
011	SUB	$S = A - B$	Soustraction
100	XOR	$S = A \oplus B$	OU exclusif
101	CMP	*Flags* uniquement	Comparaison (A-B)
110	MUL2	$S = A \times 2$	Multiplication par 2
111	DIV2	$S = A \div 2$	Division par 2

Entrées :
— A[3 :0] : Premier opérande (4 bits)
— B[3 :0] : Deuxième opérande (4 bits)
— OP[2 :0] : Code opération (3 bits)

Sorties :

— S[3 :0] : Résultat (4 bits)
— Z : Flag Zero (1 si S = 0000)
— C : Flag Carry (retenue/emprunt)
— N : Flag Négatif (bit de signe, S[3])
— V : Flag oVerflow (débordement signé)

1.2 Conception de l'unité logique (2 points)

Question 1.2.1 : Concevoir un sous-circuit `Unite_Logique` qui implémente les opérations AND, OR et XOR.
— Utiliser un multiplexeur pour sélectionner l'opération
— Documenter votre choix d'encodage pour la sélection
— Tester avec : A=1010, B=1100 pour chaque opération
Livrables :
— Sous-circuit fonctionnel dans Logisim
— Tableau des résultats de test (dans le PDF)

1.3 Conception de l'unité arithmétique (3 points)

Question 1.3.1 : Concevoir un additionneur/soustracteur 4 bits.
Rappel : La soustraction $A - B$ s'obtient par $A + \overline{B} + 1$ (complément à 2).
— Créer le sous-circuit `Add_Sub_4bits`
— Une entrée de contrôle SUB permet de choisir entre addition et soustraction
— Produire le flag Carry/Borrow en sortie
Question 1.3.2 : Implémenter MUL2 (multiplication par 2).
Indice : Un décalage à gauche équivaut à une multiplication par 2.
— $S[3] = A[2]$
— $S[2] = A[1]$
— $S[1] = A[0]$
— $S[0] = 0$
Le bit perdu (A[3]) doit activer le flag Carry.
Question 1.3.3 : Implémenter DIV2 (division entière par 2).
Indice : Un décalage à droite équivaut à une division par 2.
— $S[3] = 0$ (ou A[3] pour conserver le signe)
— $S[2] = A[3]$
— $S[1] = A[2]$
— $S[0] = A[1]$
Le bit perdu (A[0]) peut être ignoré ou signalé.
Question 1.3.4 : Créer le sous-circuit `Unite_Arithmetique` qui regroupe toutes ces opérations avec un multiplexeur de sélection.

1.4 Assemblage de l'UAL et génération des flags (2 points)

Question 1.4.1 : Assembler les unités logique et arithmétique dans un circuit principal `UAL_4bits`.
Question 1.4.2 : Implémenter la génération des flags :
— **Flag Z (Zero)** : $Z = \overline{S_3 + S_2 + S_1 + S_0}$
  — Utiliser une porte NOR à 4 entrées
— **Flag N (Négatif)** : $N = S[3]$ (bit de poids fort)
— **Flag C (Carry)** :
  — Pour ADD : retenue sortante
  — Pour SUB : emprunt (inversé)
  — Pour MUL2 : bit A[3] perdu
  — Pour autres opérations : 0
— **Flag V (Overflow)** : Détecte les débordements signés
  — Pour ADD/SUB : $V = C_{out} \oplus C_{in\_MSB}$
  — Signifie : positif+positif=négatif OU négatif+négatif=positif
Question 1.4.3 : Implémenter l'opération CMP (comparaison).
L'opération CMP effectue A-B mais ne modifie pas la sortie S. Elle met à jour uniquement les flags :
— Si A = B : Z = 1

— Si A < B : N = 1 (résultat négatif)
— Si A > B : N = 0, Z = 0
Astuce : Utiliser le soustracteur mais ignorer le résultat, conserver seulement les flags.

Partie 2 : Système de registres (4 points)

2.1 Conception d'un registre 4 bits (2 points)

Un registre est une mémoire temporaire qui stocke une valeur 4 bits.
Question 2.1.1 : Créer un sous-circuit `Registre_4bits` utilisant 4 bascules D.
Spécifications :
— **Entrées :**
  — D[3 :0] : Données à stocker
  — CLK : Horloge (front montant)
  — LOAD : Signal de chargement (1 = charger, 0 = maintenir)
  — RESET : Remise à zéro asynchrone
— **Sortie :**
  — Q[3 :0] : Valeur stockée
Indice Logisim : Utiliser le composant `Memory → Register` ou construire avec des bascules D et un multiplexeur pour le signal LOAD.
Question 2.1.2 : Tester votre registre :

1. Charger la valeur 1010
2. Vérifier qu'elle est maintenue pendant plusieurs cycles d'horloge
3. Tester le RESET

2.2 Banque de registres (2 points)

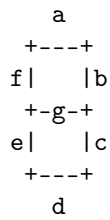
Créer un système avec 2 registres : REG_A et REG_B.
Question 2.2.1 : Concevoir un circuit `Banque_Registres` avec :
Entrées :
— DATA_IN[3 :0] : Données à écrire
— SEL : Sélection du registre (0 = REG_A, 1 = REG_B)
— LOAD : Signal d'écriture
— CLK : Horloge
— RESET : Remise à zéro globale
Sorties :
— OUT_A[3 :0] : Contenu de REG_A
— OUT_B[3 :0] : Contenu de REG_B
Question 2.2.2 : Implémenter la logique de sélection :
— Utiliser un démultiplexeur pour router le signal LOAD
— Seul le registre sélectionné doit être chargé
Tests à effectuer :

1. Charger 0101 dans REG_A
2. Charger 1100 dans REG_B
3. Vérifier que les deux valeurs sont conservées simultanément
4. Modifier REG_A sans affecter REG_B

Partie 3 : Décodeur 7 segments (3 points)

3.1 Spécifications (0.5 point)

Un afficheur 7 segments permet d'afficher les chiffres de 0 à 9 (et A à F en hexadécimal).



Les segments sont notés a, b, c, d, e, f, g.
Exemple : Pour afficher "0", activer a,b,c,d,e,f (pas g).

3.2 Table de vérité (1 point)

Question 3.2.1 : Compléter la table de vérité pour afficher les chiffres hexadécimaux 0-F.

Entrée				**Sorties (segments)**							
D_3	D_2	D_1	D_0	a	b	c	d	e	f	g	**Hex**
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0								2
0	0	1	1								3
0	1	0	0								4
0	1	0	1								5
0	1	1	0								6
0	1	1	1								7
1	0	0	0								8
1	0	0	1								9
1	0	1	0								A
1	0	1	1								b
1	1	0	0								C
1	1	0	1								d
1	1	1	0								E
1	1	1	1								F

Aide : Chercher "7 segment display truth table" pour des références visuelles.

3.3 Simplification (optionnel - bonus)

Pour les plus rapides, simplifier une ou deux sorties (segment a ou segment g) avec des tables de Karnaugh.
Note : Vous pouvez aussi utiliser directement le composant `Hex Digit Display` de Logisim, mais concevoir le décodeur vous-même rapporte plus de points.

3.4 Implémentation (1.5 point)

Question 3.4.1 : Créer le sous-circuit `Decodeur_7seg`.
Option A (recommandée) : Utiliser le composant Logisim :
— `Input/Output → Hex Digit Display`
— Connecter votre sortie 4 bits
Option B (bonus +1pt) : Implémenter manuellement le décodeur :
— Créer les 7 fonctions booléennes (a, b, c, d, e, f, g)
— Utiliser des portes logiques
— Connecter à 7 LEDs individuelles

Partie 4 : Système de détection d'erreurs (2 points)

4.1 Détecteur de débordement (1 point)

Question 4.1.1 : Créer un circuit `Detecteur_Overflow` qui allume une LED rouge si :
— Addition : résultat > 15 (flag C = 1)
— Soustraction : résultat < 0 (flag N = 1 en non-signé)
— Multiplication par 2 : bit éjecté (A[3] = 1)
Entrées :
— OP[2 :0] : Code opération
— FLAGS : Z, C, N, V
Sortie :
— ERROR : 1 si erreur détectée

4.2 Détecteur de division par zéro (1 point)

Question 4.2.1 : Ajouter une détection pour la division par 2 lorsque A = 0000 (division de 0).
Bien que mathématiquement correct ($0 \div 2 = 0$), certaines architectures signalent cette condition. Implémenter un signal `DIV_ZERO_WARNING`.
Question 4.2.2 : Créer un circuit `Gestionnaire_Erreurs` qui combine :
— Détection de débordement
— Détection de division par zéro
— Une sortie générale `ERROR` (LED rouge)
— Une sortie `WARNING` (LED jaune)

Partie 5 : Intégration finale (3 points)

5.1 Assemblage de la calculatrice (2 points)

Question 5.1.1 : Créer le circuit principal `Calculatrice_Finale` qui intègre :
— La banque de registres (REG_A et REG_B)
— L'UAL 4 bits
— Deux afficheurs 7 segments (pour REG_A et REG_B)
— Un afficheur 7 segments pour le résultat
— Le système de détection d'erreurs avec LEDs
— Les affichages des flags (4 LEDs : Z, C, N, V)
Interface utilisateur :
Entrées :
— 4 switches pour DATA_IN[3 :0]
— 1 switch SEL_REG (sélection REG_A/REG_B)
— 1 bouton LOAD (charger le registre sélectionné)
— 3 switches OP[2 :0] (sélection de l'opération)
— 1 bouton COMPUTE (lancer le calcul)
— 1 bouton RESET (tout réinitialiser)
— 1 clock (horloge pour les registres)
Sorties :
— 3 afficheurs 7 segments (REG_A, REG_B, Résultat)
— 4 LEDs pour les flags (Z, C, N, V)
— 2 LEDs d'erreur (ERROR rouge, WARNING jaune)

5.2 Scénario de test complet (1 point)

Tester votre calculatrice avec le scénario suivant et documenter les résultats :

Étape	Action	REG_A	REG_B	Résultat	Flags
1	Charger 5 dans REG_A	0101	0000	-	-
2	Charger 3 dans REG_B	0101	0011	-	-
3	ADD : A + B	0101	0011	1000	Z=0,C=0
4	Charger 15 dans REG_A	1111	0011	-	-
5	ADD : A + B	1111	0011	????	????
6	SUB : A - B	1111	0011	????	????
7	Charger 6 dans REG_A	0110	0011	-	-
8	MUL2 : A × 2	0110	0011	????	????
9	Charger 8 dans REG_A	1000	0011	-	-
10	DIV2 : A ÷ 2	1000	0011	????	????
11	XOR : A ⊕ B	1000	0011	????	????
12	CMP : comparer A et B	1000	0011	-	????

Livrables :
- Tableau complété avec tous les résultats
- Captures d'écran de 3 tests minimum
- Commentaire sur les débordements observés

Critères d'évaluation

Critère	Points	Détails
Partie 1 : UAL (8 pts)		
Unité logique	2	AND, OR, XOR fonctionnels avec MUX
Unité arithmétique	3	ADD/SUB, MUL2, DIV2 corrects
Flags	2	Z, C, N, V correctement générés
CMP	1	Comparaison fonctionnelle
Partie 2 : Registres (4 pts)		
Registre 4 bits	2	LOAD, CLK, RESET fonctionnels
Banque de registres	2	Sélection et écriture correctes
Partie 3 : Affichage (3 pts)		
Table de vérité	1	Complète et correcte
Décodeur 7 segments	1.5	Affichage hex correct (0-F)
Simplification (bonus)	+0.5	Karnaugh pour au moins 2 segments
Partie 4 : Détection erreurs (2 pts)		
Débordement	1	Détection correcte selon l'opération
Division par zéro	1	Warning implémenté
Partie 5 : Intégration (3 pts)		
Assemblage	2	Circuit complet et fonctionnel
Tests	1	Scénario complet documenté
Qualité et méthodologie		
Organisation	1	Sous-circuits bien nommés, câblage clair
Documentation	1	PDF complet avec explications
Bonus	+2	Fonctionnalités supplémentaires
TOTAL	**/20**	

Bonus possibles (max +2 points)

- **+0.5** Décodeur 7 segments manuel (sans composant Logisim)
- **+0.5** Simplification Karnaugh de 2+ segments
- **+0.5** Ajout d'une opération supplémentaire (NEG : négation en complément à 2)
- **+0.5** Affichage en décimal signé (détection et affichage du signe -)
- **+1.0** Extension à 8 bits de toute la calculatrice
- **+1.0** Ajout d'un mode pas-à-pas avec horloge manuelle

Conseils pour réussir

1. **Approche progressive :** Ne commencez pas par le circuit final. Construisez et testez chaque composant séparément.
2. **Réutilisation :** Vous pouvez vous inspirer de vos circuits du matin. C'est même encouragé !
3. **Tests unitaires :** Chaque sous-circuit doit être testé indépendamment avant intégration.
4. **Nommage clair :** Donnez des noms explicites à vos signaux et sous-circuits.
5. **Sauvegardes régulières :** Utilisez "Enregistrer sous" avec des versions (v1, v2, etc.).
6. **Documentation :** Prenez des notes au fur et à mesure. N'attendez pas la fin pour rédiger le PDF.
7. **Gestion du temps :**
   — 0h00-0h45 : Partie 1 (UAL)
   — 0h45-1h15 : Partie 2 (Registres)
   — 1h15-1h45 : Partie 3 (Affichage)
   — 1h45-2h00 : Partie 4 (Erreurs)
   — 2h00-2h30 : Partie 5 (Intégration et tests)
8. **Priorisation :** Si vous manquez de temps, concentrez-vous sur :
   (a) UAL fonctionnelle (8 pts)
   (b) Registres basiques (4 pts)
   (c) Intégration minimale (3 pts)

   Cela vous assure déjà 15/20.

Ressources autorisées

— Vos notes du TP du matin
— Le sujet du TP du matin
— Documentation Logisim : `Aide → Help`
— Tables de vérité et Karnaugh vierges (fournies)

Livrables finaux

Fichier Logisim (.circ)

— Un seul fichier contenant tous les circuits
— Circuit principal nommé "Calculatrice_Finale"
— Sous-circuits clairement nommés et organisés

Document PDF

Le PDF doit contenir :

1. **Page de garde** avec nom, prénom, date
2. **Partie 1 - UAL**
   — Schéma bloc de l'architecture globale
   — Explications des choix de conception
   — Tableau de tests de l'unité logique
3. **Partie 2 - Registres**
   — Schéma du registre 4 bits
   — Explication du système de sélection
   — Captures d'écran des tests
4. **Partie 3 - Affichage**
   — Table de vérité complète du décodeur 7 segments
   — Tables de Karnaugh (si simplification manuelle)
   — Capture d'affichage de plusieurs chiffres
5. **Partie 4 - Détection erreurs**
   — Logique de détection expliquée

   - Tests avec débordement

6. **Partie 5 - Intégration**
   - Schéma global annoté
   - Tableau de tests complet (12 étapes)
   - Au moins 3 captures d'écran légendées
   - Analyse des résultats et difficultés rencontrées
7. **Conclusion**
   - Fonctionnalités implémentées
   - Difficultés rencontrées et solutions
   - Améliorations possibles

Rendu

- **Quand ?** À la fin de la semaine
- **Comment ?** Dépôt sur la plateforme pédagogique
- **Format :**
  - 1 fichier .circ
  - 1 fichier PDF
  - Les deux dans une archive .zip nommée : `NOM_Prenom_ProjetLogisim.zip`

Bon courage !
N'hésitez pas à appeler l'enseignant si vous êtes bloqué sur un point technique.
La méthodologie et la démarche sont aussi importantes que le résultat final.