Un simulateur logique pensé pour l'enseignement

Jean-Philippe PELLET

Haute École Pédagogique du canton de Vaud, Lausanne, Suisse

Numéro thématique 2 / 2023 - T3

RÉSUMÉ *Une des plus-values d'un enseignement appliqué de l'informatique est de pouvoir amener les élèves à construire de petits systèmes informatiques avec lesquels on peut interagir. On pense par exemple à la programmation d'une interface graphique, d'un petit robot ou d'un système de carte électronique comme un Microbit ou Arduino. Dans cet atelier, présenté lors du colloque Didapro 9 en 2022 au Mans, nous descendons d'un niveau dans les couches matérielles tout en restant dans le même état d'esprit en proposant un outil de simulation de portes logiques et d'autres composants informatiques de base, que nous développons. Nous montrons comment ce simulateur se distingue d'autres solutions par ses possibilités pédagogiques bien appropriées à une approche PRIMM (Predict-Run-Investigate-Modify-Make), une extension du modèle bien connu Use-Modify-Create. Nous proposons dans l'atelier quelques scénarios pédagogiques adaptés à des élèves de début de lycée.*

MOTS-CLÉS • *Enseignement de l'informatique, lycée/gymnase, simulateur logique, architecture des ordinateurs*

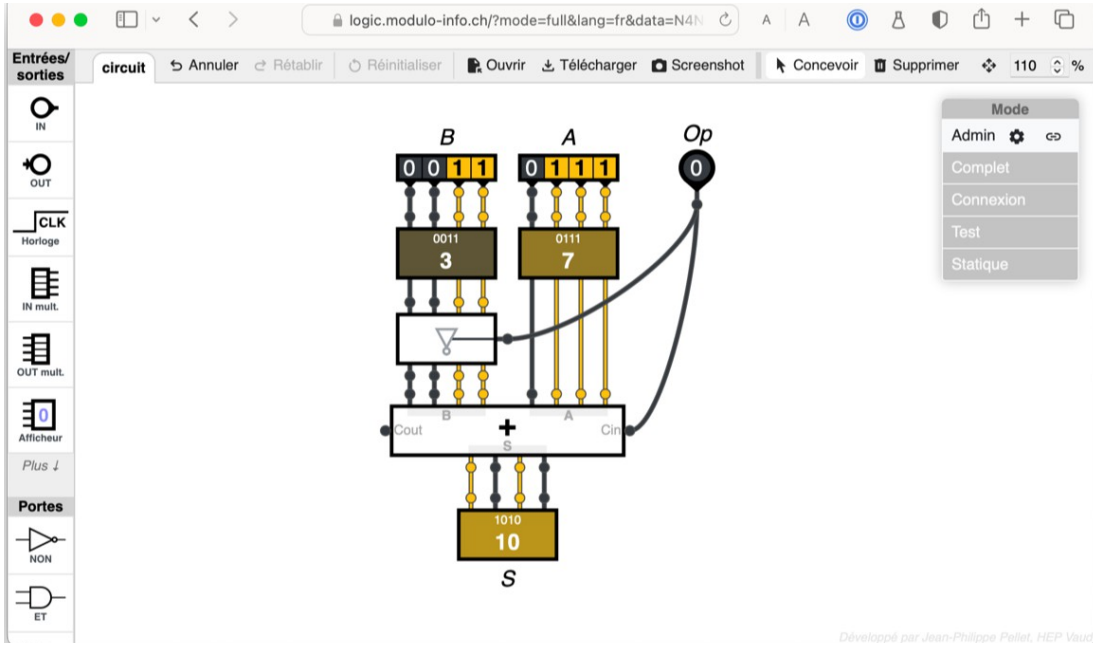

Figure 1 : L'interface du simulateur logique

Objectif

Un pan de l'informatique quelquefois absent dans la plupart des cours qui nous sont connus traite de l'architecture de base des ordinateurs, ou plus précisément des systèmes logiques et de composants de base comme une unité arithmétique et logique (ALU) ou une bascule. Nous pensons qu'il y a quelques moments pédagogiques très intéressants à faire vivre au sein de cette thématique dès le lycée. En particulier, la réalisation de petits circuits montrant « *comment calculer avec de l'électricité* » ou « *comment stocker des données avec de l'électricité* », pour vulgariser la formulation, semble abordable avec des supports pédagogiques appropriés.

Nous pensons que l'inspection de telles situations problèmes est un moyen pour les apprenants de construire une meilleure représentation mentale de comment fonctionne un ordinateur, d'expérimenter concrètement de petits systèmes « entrée–traitement–sortie », et de réaliser comment une abstraction matérielle progressive nous permet de finalement construire un mini-système complet.

Le simulateur logique que nous avons développé à cet effet permet des mises en œuvre pédagogiques que nous pensons bien plus riches (et techniquement plus simples) que ce qui existe actuellement comme solution logicielle dans le contexte d'une introduction à l'informatique.

Description de l'atelier et la ressource

On retrouve souvent les thématiques d'architecture des ordinateurs, des processeurs ou des systèmes logiques dans les études tertiaires. Obtenir le matériel nécessaire à grande échelle pour aborder ces thématiques est un problème en termes de coûts, de fiabilité et de flexibilité du matériel pédagogique. Nous avons exploré des options comme les *logidules* (cf. liens ci-dessous), qui semblent particulièrement adaptés, mais ne sont malheureusement plus en production. Nous avons ainsi cherché à utiliser, dans un premier temps en tout cas, un simulateur afin de faciliter l'utilisation en classe.

Plusieurs solutions existent; nous en mentionnons deux en particulier, qui nous semblaient initialement prometteuses mais typiques de leur non-adéquation pour nos besoins :

- *Logisim* est un logiciel open source de simulation de circuits logiques régulièrement utilisé. S'il est open source, Logisim a été créé il y plus de 20 ans, repose sur des technologies de son époque et marque son âge dans les instructions d'installation et l'interface.
- *Logicly*, plus moderne et basé sur une solution web, a l'avantage de ne requérir aucune installation, mais fonctionne selon un modèle freemium, que nous ne jugions pas désirable en classe, même s'il a été mis en œuvre par d'autres dans des contextes comparables.

Nous nous sommes ainsi lancés dans le développement de notre propre solution (basée sur un simulateur open source existant, mais finalement intégralement réécrit et enrichi de nombreuses fonctionnalités.

L'approche PRIMM

Nous voulions explicitement que notre solution soit simple à mettre en œuvre via plusieurs supports ou plateformes que les enseignants seraient susceptibles d'utiliser, et puisse activement prendre en charge une approche de type *Predict-Run-Investigate-Modify-Make* (PRIMM).

PRIMM peut être vu comme une extension du modèle bien connu *Use-Modify-Create* (UMC) (Lee et al., 2011). Sentance et Waite (2017) expliquent que ces modèles découpent en plusieurs étapes bien définies l'approche pédagogique de ce que nous appellerons un *processus de synthèse structurée*. Les apprenants sont ainsi invités à d'abord (a) faire tourner des systèmes déjà réalisés ; (b) effectuer des modifications sur un système existant ; avant de (c) créer un système soi-même. Pour faire simple, la différence entre PRIMM et UMC est une spécification plus détaillée de la première étape (a).

PRIMM décompose en effet la première étape Use de UMC en trois étapes : Predict, Run et Investigate. Les étapes (b) et (c) peuvent être considérées comme identiques dans les deux approches.

Ces approches ont été d'abord développées avec en tête un contexte d'apprentissage de la programmation, mais le cas de la création de circuits logiques et de la compréhension du fonctionnement de briques de base à disposition est conceptuellement très proche. Il s'agit dans les deux cas de combiner de façon structurée des «primitives» dont le comportement est bien défini afin de résoudre un problème.

Caractéristiques du simulateur

Voici les caractéristiques du simulateur qui soutiennent une approche PRIMM et dont certaines sont exclusives à notre solution.

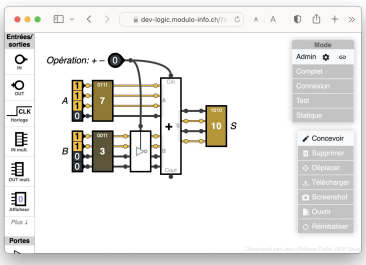

Figure 2: Utilisation directe en mode Édition, construction d'un soustracteur 4 bits en utilisant un additionneur

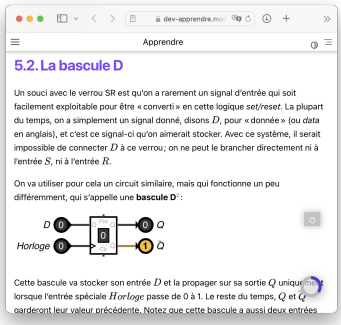

Figure 3: Intégration dans une page web en mode Test d'un circuit

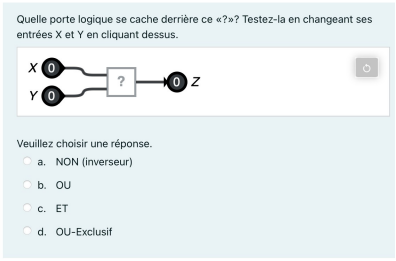

Figure 4: Intégration dans une question Moodle avec une partie du circuit cachée

Tout d'abord, à toutes les étapes du processus PRIMM sauf la dernière (Make), il est essentiel que les enseignants puissent diffuser des schémas logiques partiels ou complets avec lesquels les apprenants pourront interagir ou qu'ils seront en mesure de modifier. Pour prendre en charge les différentes étapes de PRIMM, nous avons fait en sorte que le simulateur dispose de plusieurs modes d'interaction : du mode le plus simple, qui va d'un circuit statique, au mode enseignant, qui permet de composer un nouveau circuit en avec des composants personnalisés voire volontairement défectueux, en passant par le mode où l'on peut uniquement changer les entrées d'un circuit dont les connexions sont préétablies.

Nous avons opté pour une solution web, qui ne requiert aucune installation. De plus, des paramètres d'URL dans les liens partagés peuvent contenir la description complète d'un schéma

logique. Par exemple, ce lien (cliquer ici) donne directement un accès complet au schéma exemple représenté sur la Fig. 2.

Aucune base de données n'est requise et aucun stockage ne doit être effectué côté serveur; aucun compte ou profil ne doit être créé, ni pour les enseignants, ni pour les apprenants. La solution est donc particulièrement légère à mettre en place en termes de gestion d'une classe numérique.

Voici comment les étapes de PRIMM peuvent être mises en œuvre avec notre simulateur :

- *Predict*. Des circuits peuvent être mis à disposition des apprenants en mode «inerte» pour que les apprenants fassent leurs propres prédictions sur leur comportement sans utiliser le simulateur. Les entrées et sorties peuvent être affichées de manière neutre (sans révéler la valeur véhiculée). Le contenu des composants mémoire (verrous, bascules, registres, etc.) peut être masqué.
- *Run*. Des circuits peuvent être diffusés en mode lecture seule, mais en laissant les apprenants changer les valeurs d'entrées pour observer le comportement (et éventuellement autovalider des hypothèses faites à l'étape *Predict*).
- *Investigate*. Des circuits peuvent être volontairement créés comme défectueux (fausse fonction logique réalisée ; sortie bloquée sur une valeur fixe ; temps de propagation trop élevé ; porte logique qui s'affiche selon une fonction donnée mais en fait en réalise une autre, etc.) pour placer l'apprenant dans une position d'investigation de ce qui ne fonctionne pas correctement. Pour les trois cas de figure précédents (P, R, I), le simulateur est mis dans un mode qui ne permet pas d'ajouter des composants ou d'en modifier le comportement.
- *Modify et Make*. Les outils d'édition sont complètement disponibles sans restriction pour les apprenants, qui peuvent démarrer de zéro ou depuis un début de circuit partagé par l'enseignant en mode édition.

Par ailleurs, une série de facilités techno-pédagogiques rendent le simulateur spécialement flexible :

- Le simulateur peut être utilisé directement depuis son site hôte (voir fig. 2), mais peut être intégré dans n'importe quelle page web (fig. 3 et 4), soit via un élément `<iframe>` pour un maximum de compatibilité, soit avec un élément `<logic-editor>` mis à disposition via l'inclusion d'un script *JavaScript* unique. Il est également possible d'interagir via *JavaScript* avec le simulateur pour mettre en évidence une partie d'un circuit, vérifier l'état des entrées et sorties d'un composant, sauvegarder ou charger des données, etc.
- Les liens générés contenant la structure d'un circuit et les éditeurs inclus dans des pages web externes peuvent choisir de mettre à disposition un nombre restreint de composants dans la «bibliothèque». En effet, avoir la totalité des composants visuellement disponibles est de nature à créer une surcharge visuelle, voire cognitive. Ainsi, on peut par exemple demander de créer une porte de type OU-Exclusif sans mettre cette dernière à disposition, ou encore de concevoir un circuit permettant de faire l'addition ou, à choix, la soustraction de deux nombres d'entrées sur la base d'un composant effectuant uniquement l'addition et dans disposer d'une ALU complète. Il est également possible de prédisposer sur le canevas de travail une série de composants déjà instanciés et de demander de les connecter uniquement.

- La propagation des valeurs sur un circuit est animée. Ceci permet non seulement de voir les valeurs logiques se propager, mais aussi de concevoir la limite de fréquence de circuits même simples. Par exemple, le changement d'entrées de 4 additionneurs en cascade qui cause la propagation d'une retenue sur plusieurs colonnes (0111 + 0001 par exemple) montre à la fois la logique derrière l'enchaînement successif des composants et, en faisant apparaître temporairement sur les sorties des valeurs intermédiaires qui ne représentent pas le résultat de l'addition, démontre que la montée en fréquence d'un tel circuit est limitée.
- Des *tooltips* (infobulles) expliquent le fonctionnement des composants disponible. Non seulement de manière générique, mais aussi en fonction de leur état ou de leurs entrées. Par exemple, le *tooltip* d'une porte logique explique la sortie avec la ligne correspondante de sa table de vérité; le *tooltip* d'une bascule indiquera son état interne et ce qui se passera au prochain coup d'horloge. Des étiquettes textuelles peuvent aussi être placées dans le circuit et leur contenu peut dépendre de la valeur d'une entrée, d'une sortie, d'un afficheur, etc.
- L'investigation par les apprenants de circuits volontairement défectueux, conçus comme tels par un enseignant, est à notre avis spécialement intéressante et caractéristique de ce que l'informatique dans son aspect « science de l'ingénieur » a à apporter. En effet, il est rare, en pratique, que des composants réels reproduisent parfaitement et indéfiniment leur comportement théorique idéal. Par ailleurs, des activités telles que le développement de techniques d'investigation, la recherche systématique et l'isolement de la source d'un problème sont de bons exercices d'un point de vue d'entraînement de la pensée computationnelle.

Une série d'exemples mettant en œuvre ce qui a été décrit ici a été présenté lors de l'atelier de Didapro 9 et se trouve en ligne à cette adresse: https://jp.pellet.name/hep/didapro9/.

Expérimentations réalisées et liens avec la recherche

La création de notre simulateur logique s'est inspiré des travaux de recherche suivant :

- Logidules: Nicoud, J.D.(1991). Dedicated tools for microprocessor education. IEEE Micro 11(1), p. 14–17.
- Logisim: A graphical system for logic circuit design and simulation. Journal on Educational Resources in Computing (JERIC) 2(1), 5–16 (2002)
- Utilisation de Logisim dans l'éducation: Luković, V., Krneta, R., Vulović, A., Dimopoulos, C., Katzis, K., Meletiou- Mavrotheris, M.(2016). Using Logisim educational software in learning digital circuits design. In : Proceedings of 3rd International Conference on Electrical, Electronic and Computing Engineering IcETRAN. p. 5–1.
- Utilisation de Logic.ly dans l'éducation: Rueda, R.A.S., Silis, J.A.S. (2018). Logic.ly simulator. Technological tool to facilitate the teaching-learning process about mathematics ? Dilemas Contemporáneos : Educación, Política y Valore (3).
- Use–Modify–Create: Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads 2(1), p. 32–37.
- PRIMM: Sentance, S., Waite, J. (2017). PRIMM: Exploring pedagogical approaches for teaching text- based programming in school. In: Proceedings of the 12th Workshop on Primary and Secondary Computing Education. p. 113–114.

Liens vers les ressources et point de contact

- Le simulateur logique est disponible sur ce site : https://logic.modulo-info.ch
- Cette activité a été développée dans le cadre de la réforme de l'enseignement de l'éducation numérique du canton de Vaud en Suisse
- Jean-Philippe Pellet, Haute École Pédagogique du canton de Vaud, Lausanne, Suisse (jean-philippe.pellet@hepl.ch).