HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Laboratoire VSN

semestre de printemps 2020 - 2021

Récepteur Morse

Composant à tester

Nous souhaitons tester un système permettant la réception de code Morse pour les caractères alphanumériques standards. Le composant est chargé d'analyser une ligne série sur laquelle sont envoyés des codes Morse, et d'en reconstruire les codes ASCII correspondants.

Paramètres génériques

Le composant possède un paramètre générique `LOG_RELATIVE_MARGIN`, qui définit la tolérance sur la longueur des dots et dashes.
Un deuxième paramètre générique, `ERRNO`, a une fonctionnalité expliquée plus loin dans ce document.

Entrées/sorties

L'entité est déclarée ainsi :

```
entity morse_char_receiver is
    generic (
        LOG_RELATIVE_MARGIN : integer := 0;
        ERRNO               : integer := 0
    );
    port (
        clk_i              : in  std_logic;
        rst_i              : in  std_logic;
        char_o             : out std_logic_vector(7 downto 0);
        char_valid_o       : out std_logic;
        unknown_o          : out std_logic;
        dot_period_error_o : out std_logic;
        morse_i            : in  std_logic;
        dot_period_i       : in  std_logic_vector(27 downto 0)
    );
end morse_char_receiver;
```

Les entrées/sorties sont :

Nom	Taille	Dir	Description
`clk_i`	1	in	Horloge du système
`rst_i`	1	in	Reset du système
`char_o`	8	out	Valeur ASCII du caractère reçu, sur 8 bits
`char_valid_o`	1	out	Indique qu'un caractère est valide, et se trouve sur `char_o`
`unknown_o`	1	out	Indique qu'un code Morse non reconnu a été reçu
`dot_period_error_o`	1	out	Indique que la ligne est passée à '1' pendant un temps non légal
`dot_period_i`	28	in	Nombre de cycles d'horloge correspondant à la durée d'un *dot*
`morse_i`	1	in	Ligne Morse en entrée

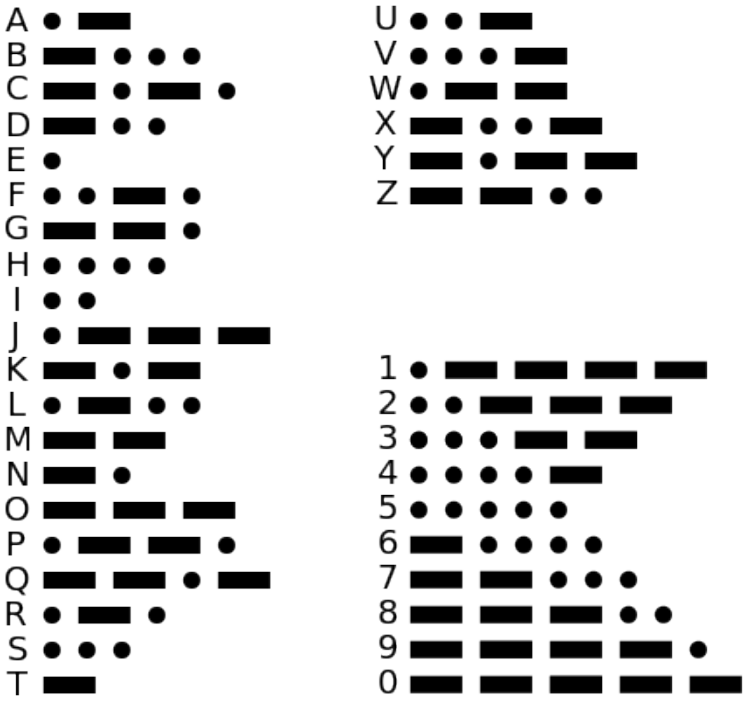
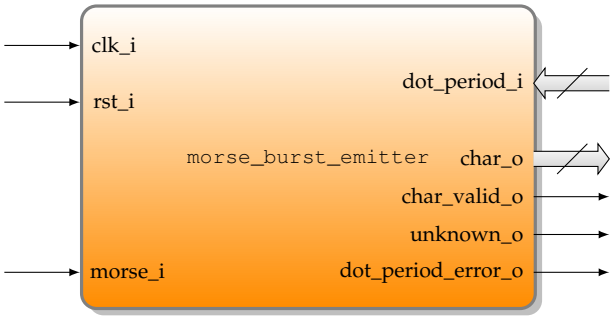


FIGURE 1. *Code morse standard (Source : wikipedia)*



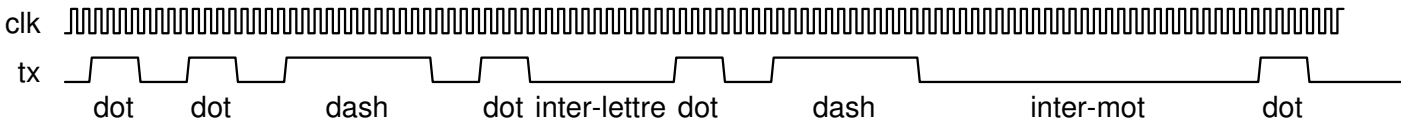
Fonctionnement

Le code morse permet d'envoi d'information en codant les lettres grâce à des activations d'un signal sur des durées longues ou courtes. Nous ne traiterons ici que des caractères standards, à savoir toutes les lettres (minuscule et majuscule), et les chiffres. Nous ajouterons également l'espace afin de pouvoir séparer des mots, ainsi que le *carriage return* (retour à la ligne).
La figure 1 présente la représentation usuelle des caractères qui nous intéressent [1].
Nous utiliserons le terme *dot* pour désigner le point, et par abus de langage la durée d'un point. Le terme *dash* représentera un tiret, et sa durée associée. Les durées sont les suivantes :

Signal Morse	durée (en *dot*)	Description
'1'	1	Un *dot* codant pour une partie de lettre
'1'	3	Un *dash* codant pour une partie de lettre
'0'	1	Un espace entre deux *dot* ou *dash*, à l'intérieur d'une lettre
'0'	3	Un espace délimitant deux lettres
'0'	7	Un espace délimitant deux mots

1. Il existe plusieurs standards

Si le signal est à '0' pendant 7 *dot*, nous considérerons ceci comme un seul espace, mais s'il est à '0' pendant plus de 7 *dot*, alors nous considérerons qu'il s'agit d'un retour à la ligne, correspondant au caractère ASCII 13.
Le chronogramme suivant présente l'envoi de la lettre F suivie de A, d'un espace et de la lettre E, avec une durée de quatre cycles d'horloge par *dot*.



Lorsque le récepteur détecte un code Morse valide, il doit activer la sortie `char_valid_o` pendant un cycle d'horloge, et au même instant la sortie `char_o` doit contenir la valeur ASCII du caractère détecté. Pour les lettres, la sortie doit correspondre au code ASCII de la majuscule correspondante. Outre les lettres et les nombres, l'espace (code 32) et le CR (code 13) seront des valeurs valides.
Si le composant détecte un code Morse non supporté il doit activer la sortie `unknown_o` pendant un cycle d'horloge.
Si le composant détecte que la ligne est à '1' pendant un temps non légal, il doit activer la sortie `dot_period_error_o` pendant un cycle d'horloge.
Etant donné qu'il est impossible que l'émetteur soit exactement synchrone avec le récepteur, celui-ci peut fonctionner avec une certaine marge d'erreur. En théorie l'entrée `dot_period_i` définit la durée d'un dot. Toutefois le paramètre générique permet de définir une marge à appliquer.

LOG_RELATIVE_MARGIN	**marge**
0	0
1	$\lfloor dot\_period/2 \rfloor$
2	$\lfloor dot\_period/4 \rfloor$
3	$\lfloor dot\_period/8 \rfloor$
X	$\lfloor dot\_period/2^X \rfloor$

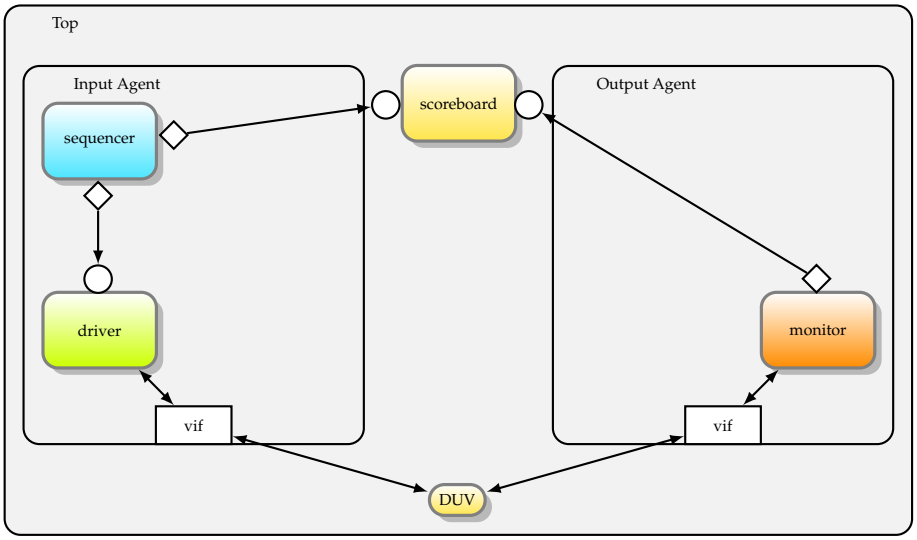
On considère alors l'intervalle $[dot\_period - marge, dot\_period + marge]$ pour détecter un dot valide ou un espace valide. Le dash sera détecté via l'intervalle $[3(dot\_period - marge), 3(dot\_period + marge)]$, et le même concept sera appliqué pour les espaces inter-dot, inter-lettres et inter-mots.
La version du DUV qui vous est fournie est parfaitement fonctionnelle, toutefois un paramètre générique `ERRNO` permettra d'injecter artificiellement des erreurs dans le design. Il s'agira d'un entier qui offre le comportement suivant :

1. S'il vaut 0, le résultat est valide ;
2. S'il est compris entre 2 et 11, le résultat n'est pas valide.

Ce paramètre générique vous permettra de valider votre banc de test, en le simulant avec toutes les valeurs de `ERRNO`.
⚠ L'entité à tester ne doit pas être modifiée. Elle est encryptée, ce serait de toute façon un peu difficile.

Architecture

Le banc de test a été décomposé selon la méthodologie vue en cours, afin de bien différencier les responsabilités (séquenceur, driver, moniteur, scoreboard). La structure globale vous est déjà fournie, avec les instanciations de composants et les connexions via des FIFOs. Passez un peu de temps à bien comprendre cette architecture en regardant les fichiers SystemVerilog.



Vous pouvez noter les points suivants :

1. Les transactions envoyées par l'agent Input au scoreboard viennent du séquenceur plutôt que d'un moniteur. Dans notre cas précis cela permettra de faciliter la détection de certaines erreurs.
2. Il n'y a pas de driver de sortie, car rien n'est en entrée du DUV si ce n'est le `dot_period_i`. Pour ce signal, c'est le driver d'entrée qui est chargé de le définir, de manière à faciliter la mise en place des scénarios.

Un banc de test fonctionnel vous est fourni. Il est compilable et simulable, mais comme vous le verrez il n'offre pas de traitement réellement pertinent. A vous de modifier les transactions/séquenceur/driver/scoreboard afin de pouvoir correctement vérifier le DUV. Le moniteur de sortie est déjà fonctionnel.
Il est suggéré de travailler dans l'ordre suivant :

1. Modification du driver
2. Modification du scoreboard
3. Modification du séquenceur et des transactions

Le script de simulation peut être lancé depuis le répertoire `sim` (exemple ici, pour le testcase 1, taille de `LOG_RELATIVE_MARGIN` à 2, et ERRNO à 0) :
`do ../scripts/sim.do all 2 1 0`
Pour la modification du séquenceur et des transactions, il est évidemment nécessaire de réfléchir aux scénarios de test. Un testplan vous est fourni. Il contient déjà les lignes correspondantes aux tests des différents ERRNO. Ajoutez vos propres scénarios de test. N'hésitez pas à stresser un peu le système (ou non).
Le testbench a un paramètre générique `TESTCASE`, qui vise à pouvoir choisir le testcase à jouer. Utilisez ce paramètre. Afin de pouvoir automatiser les tests, faites que le testcase 0 lance tous les testcases à la suite.
Un fichier `default.rmdb` vous est fourni. Vous pouvez lancer `vrun directed` pour valider votre système. Les simulations seront lancées avec des tailles différentes, des `ERRNO` différents, et sur les deux designs. Il est également pertinent d'y ajouter les testcases que vous avez décidé d'implémenter, afin de lier votre testplan au fichier rmdb.

Travail à rendre

— Générez une archive du projet à l'aide du script `vsn_rendu.sh`. Ce script vous est fourni avec le code et l'énoncé du laboratoire. Il génère une archive pour autant que le rapport (fichier `rapport.pdf`) et le fichier testbench soient présents. Attention à lancer le script depuis le répertoire où il se trouve. Il faut donc que le script se trouve dans le même répertoire que le fichier `rapport.pdf` et que le répertoire `code`.
— Copiez l'archive sur Cyberlearn.

Barème de correction

Documentation	20%
Pertinence des testcases	20%
Architecture du code	15%
Détection / non détection correcte	25%
Codage	10%
Commentaires au niveau du code	10%