|  |
| --- |
|  |
| **Laboratoire 6**  **Mesure du temps de réaction** |
| par :  Patrick Maillard  Guillaume Gonin  pour :  Etienne Messerli (Professeur)  Anthony Convers (Assistant) |
| HEIG-VD  Département TIC  27.01.2025 |

Table des matières

[UART 3](#_Toc188869287)

[Analyse 3](#_Toc188869288)

[Conception 3](#_Toc188869289)

[Test 4](#_Toc188869290)

[Emetteur série 5](#_Toc188869291)

[Analyse 5](#_Toc188869292)

[Conception 5](#_Toc188869293)

[Test 6](#_Toc188869294)

[Interruption 7](#_Toc188869295)

[Analyse 7](#_Toc188869296)

[Conception 7](#_Toc188869297)

[Test 8](#_Toc188869298)

[Compteur 9](#_Toc188869299)

[Analyse 9](#_Toc188869300)

[Conception 9](#_Toc188869301)

[Test 9](#_Toc188869302)

[Nombre aléatoire 10](#_Toc188869303)

[Analyse 10](#_Toc188869304)

[Conception 10](#_Toc188869305)

[Test 10](#_Toc188869306)

[Application principal 10](#_Toc188869307)

[Analyse 10](#_Toc188869308)

[Conception 11](#_Toc188869309)

[Test 11](#_Toc188869310)

[Emetteur Série UART 12](#_Toc188869311)

[Question 12](#_Toc188869312)

[Réponse 12](#_Toc188869313)

[Partie VHDL 12](#_Toc188869314)

[Schémas blocs 13](#_Toc188869315)

[Simulation 15](#_Toc188869316)

[Conclusion 16](#_Toc188869317)

[Annexes 17](#_Toc188869318)

# UART

## Analyse

Pour la partie UART il est important de trouver des différentes valeurs dont on aura besoin afin de pouvoir communiquer avec la carte DE1-SoC en UART. Pour ce faire nous aurons besoin de trouver l’adresse de base du module UART0. Le voici ci-dessous :

* Adresse de base : **0xFFC02000**

Il nous est aussi demandé de configurer l’UART de la manière suivante :

* Baudrate : 9600
* Bit de données : 8
* Bit de parité désactivé
* Bit de stop : 1
* Activer les buffers FIFO en émission et réception

Nous devons donc setter certain registre correctement pour le module UART0, ces registres sont les suivant :

* rbr\_thr\_dll (dll), ier\_dlh (dlh) : ces 2 registre nous permettrons de setter la valeur du diviseur nous permettant de définir un baudrate où dlh représente la partie MSB de ce diviseur et dll la partie LSB. (Calcul de la valeur du diviseur => divisor = l4\_sp\_clk/ (16\*baudrate) =~ 651.04). ier\_dlh doit être mise à 0 entièrement une fois le diviseur set.
* Lcr : ce dernier permet d’enable/disable l’écriture et la lecture du diviseur donc des dll et dlh. Nous pouvons aussi setter le nombre de bit de donnée, si l’on enable/disable la parité et si l’on enable/disable le bit de stop
* Fcr : le registre fcr permet d’activer/désactiver la FIFO, ainsi que de setter différent paramètre de la fifo (ces paramètres seront laissés par défaut)

Une fois la configuration de l’UART faites il nous faudra effectuer l’envoie de donnée. Il faut se rappeler que l’uart envoie des données en série et qu’il n’enverra les infos que caractère par caractère. Pour envoyer des données nous aurons besoin des registres suivant :

* rbr\_thr\_dll (thr) : ce sera dans ce registre que les caractères seront envoyés
* Lsr : un des bits de ce registre nous permet de vérifier si notre FIFO est vide, il nous aidera donc à n’envoyer des données que si la FIFO n’est pas vide.

## Conception

Après cette analyse nous nous retrouvons donc avec ce code C pour l’initialisation de l’uart :

void uart\_config(){

uint32\_t \*uart\_lcr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + LCR));

uint32\_t \*uart\_fcr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + FCR));

uint32\_t \*uart\_dll = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + RBR\_THR\_DLL));

uint32\_t \*uart\_dlh = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + IER\_DLH));

uint32\_t \*uart\_ier = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + IER\_DLH));

//configure le DLAB de lcr

\*uart\_lcr |= 0x80;

// divisor = l4\_sp\_clk/ (16\*baudrate) =~ 651.04

// Configurer le baud rate (651 = 0x28B)

\*uart\_dll = 0x8B; // LSB

\*uart\_dlh = 0x02; // MSB

// Configurer LCR : 8 bits de données, parité désactivée, 1 bit de stop

\*uart\_lcr = 0x03;

// Activer les FIFO en réception et transmission

\*uart\_fcr = 0x01;

// clear DLAB of lcr

\*uart\_ier = 0x00;

}

Et nous retrouvons avec ce code pour l’envoie des donnée (caractère par caractère comme cité plus haut) :

void uart\_send\_char(char c) {

uint32\_t \*uart\_thr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + RBR\_THR\_DLL));

uint32\_t \*uart\_lsr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + LSR));

while (!(\*uart\_lsr & LSR\_THRE\_MASK));

\*uart\_thr = c;

}

void send\_to\_uart(const char\* mystring){

while (\*mystring) {

uart\_send\_char(\*mystring);

mystring++;

}

}

## Test

|  |  |
| --- | --- |
| Description | OK/KO |
| Des messages de 1 caractère s’affiche dans la console | OK |
| Des messages de plusieurs s’affiche dans la console | OK |

# Emetteur série

## Analyse

L’émetteur série est très simple à mettre en place du côté du C, en effet comme expliqué dans la partie vhdl, c’est ce dernier qui mettra en place la logique principale de l’envoie à la Max10, bit par bit. Dans le C je dois donc vérifier le statut de la Max10, écrire à la bonne adresse la valeur que je souhaite envoyer mes données, ainsi que vérifier si un envoie est déjà en cours à l’aide de f\_ready\_s.

## Conception

La fonction ci-dessous vérifie l’état de f\_ready\_s et renvoi true si l’émetteur est prêt pour faire un envoie

bool serial\_transmitter\_ready(){

volatile uint32\_t \*ready = SERIAL\_LINK\_READY\_REG;

uint32\_t reg\_val = \*ready;

return ((reg\_val & SERIAL\_LINK\_READY\_MASK) == SERIAL\_LINK\_READY\_MASK);

}

La fonction suivante consiste simplement à éteindre entièrement toute les LED de la Max10, pour ce faire je vais écrire à chaque code (une sorte d’adresse pour les LED de la Max10) la valeur 0. Bien évidemment avant d’envoyé, il nous faut vérifier que nous pouvons bel est bien écrire sur la ligne série avant un quelconque envoie de donnée.

void Max10\_init(void){

uint32\_t code = 0x0;

uint32\_t value = 0x0;

volatile uint32\_t \*max10\_data\_reg = SERIAL\_LINK\_DATA\_REG;

while(code <= 0xA){

if(serial\_transmitter\_ready()){

uint32\_t myserial = code << SERIAL\_LINK\_CODE\_SHIFT | value << SERIAL\_LINK\_DATA\_SHIFT;

\*max10\_data\_reg = myserial;

code++;

}

}

uint32\_t myserial = code << SERIAL\_LINK\_CODE\_SHIFT | value << SERIAL\_LINK\_DATA\_SHIFT;

\*max10\_data\_reg = myserial;

}

La 3ème fonction vient juste récupérer la valeur du status pour vérifier que notre Max10 est connecté correctement

bool Max10\_check\_status(void)

{

volatile uint32\_t \*status = STATUS\_REG;

return ((\*status & STATUS\_VALID) == STATUS\_VALID);

}

Cette fonction reprend le code et la valeur qu’on lui envoie, le réécrit en tant qu’une seule valeur et l’envoie à la Max10.

void Max10\_write\_serial\_link(uint32\_t value, uint8\_t code)

{

volatile uint32\_t \*max10\_data\_reg = SERIAL\_LINK\_DATA\_REG;

if (code > 0x0A)

{

printf("Erreur : Valeur de sel invalide.\n");

return;

}

while(!serial\_transmitter\_ready()){}

uint32\_t myserial = code << SERIAL\_LINK\_CODE\_SHIFT | value << SERIAL\_LINK\_DATA\_SHIFT;

\*max10\_data\_reg = myserial;

}

Comme l’écriture du carré de LED se fait en 2 fois nous devons envoyer 2 trames.

void Max10\_write\_square(uint32\_t value){

uint32\_t value\_MSB = value >> 16;

uint32\_t value\_LSB = value & 0xFFFF;

Max10\_write\_serial\_link(value\_MSB, 0x7);

Max10\_write\_serial\_link(value\_LSB, 0x6);

}

## Test

|  |  |
| --- | --- |
| Description | OK/KO |
| L’envoie d’une trame sans que f\_ready\_s ne soit actif met en attente l’envoi du message | OK |
| L'envoie d’une seule trame pour les codes 0000 à 1010 (appelle à Max10\_write\_serial\_link) allume et/ou éteint les LED choisie | OK |
| L'envoie d’une seule trame pour le carré de LED allume et/ou éteint les LED choisie (appelle a Max10\_write\_square) | OK |
| Le registre de statut de la Max10 nous retourne bel et bien un statut de 01 lorsque la carte est connectée correctement | OK |

# Interruption

## Analyse

La partie interruption consiste principalement à init correctement les interruptions, pour ce faire nous allons tout simplement nous servir de l’exemple fourni dans la datasheet [ci.](https://reds.heig-vd.ch/share/cours/SoCF/gic_altera_manual_short.pdf) Afin de compléter correctement l’exemple de la datasheet il nous faut aussi trouver le numéro d’interruption qui nous intéresse, dans notre cas il s’agit tout simplement de l’IRQ 72 (FPGA\_IRQ0).

Cette IRQ est déclenché lorsque l’on a une pression sur le key0 (défini dans le vhdl) nous devons faire une fonction handler qui est appelé lorsque l’interruption est levée. Ce dernier effectuera un toggle de la Led 9, un clear de l’IRQ ainsi que récupérer le temps qu’il a fallu à l’utilisateur entre le démarrage du temps de réaction et la pression de key 0 et de l’afficher à l’aide de l’UART. Si key0 est pressé trop tôt, alors un message d’erreur est envoyé à l’UART et un compteur d’erreur est incrémenter.

## Conception

On peut voir dans ce code que lorsque l’on a le numéro d’interruption 72 qui est levé, on peut exécuter notre handler fpga\_ISR. Dans cette dernière nous faisons un clear de l’IRQ (écriture à 1 dans le registre du clear IRQ) nous faisons un toggle de la LED9 puis nous appelons une fonction appelé new\_time qui est expliqué plus bas.

void fpga\_ISR(void){

new\_time();

Leds\_toggle(LED9);

clear\_irq();

}

// Define the IRQ exception handler

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_irq (void)

{

// Read CPU Interface registers to determine which peripheral has caused an interrupt

int interrupt\_ID = \*((int \*)0xFFFEC10C);

// Handle the interrupt if it comes from the fpga

if (interrupt\_ID == 72) // check if interrupt is from the KEYs (FPGA\_IRQ0)

fpga\_ISR();

else

while (1); // if unexpected, then stay here

// Clear interrupt from the CPU Interface

\*((int \*)0xFFFEC110) = interrupt\_ID;

return;

}

Dans cette fonction nous venons tous simplement récupérer la valeur actuelle du compteur qui est notre temps de réaction, il est ensuite comparé au meilleur et à la pire réaction et remplacera l’un des 2 si l’un des 2 critère est vrai. Si le début de la mesure du temps de réaction n’a pas été fait et que l’on a pressé key0 alors un message d’erreur est envoyé à l’UART et un compteur d’erreur est incrémenter.

Et pour finir un message et envoyé à l’UART pour être affiché dans le terminal, donnons tout ce qu’il s'est passé depuis que le programme a été lancé

void new\_time(){

if(game\_started){

my\_reaction\_time = (float)counter\_current\_value() \* (float)0.000000020;

if(best\_reaction\_time > my\_reaction\_time){

best\_reaction\_time = my\_reaction\_time;

}

if(worst\_reaction\_time < my\_reaction\_time){

worst\_reaction\_time = my\_reaction\_time;

}

// display symbole de fin

Max10\_write\_square(END\_DISPLAY);

}

else{

send\_to\_uart("Error: Premature key press detected.\r\nMeasurement aborted.\r\n");

nbr\_of\_error++;

error = true;

// display off

Max10\_write\_square(0x0);

}

char buffer[256]; // Create a buffer to hold the formatted string

sprintf(buffer,

"Reaction time: %f s\r\n"

"Best time: %f s\r\n"

"Worst time: %f s\r\n"

"Errors: %d\r\n"

"Total attempts: %d\r\n",

my\_reaction\_time, best\_reaction\_time, worst\_reaction\_time, nbr\_of\_error, nbr\_of\_attempt);

send\_to\_uart(buffer);

disable\_counter();

game\_started = false;

}

## Test

|  |  |
| --- | --- |
| Description | OK/KO |
| La pression sur key0 appelle bel est bien le handler et donc toggle la LED9 et clear l’IRQ (new\_time pas pris en compte car cela est testé lors de la mise en place complète du programme principal) | OK |
| Multiple pression sur key0 toggle plusieurs fois la LED9 | OK |
| Aucune autre interaction que la pression sur key 0 n’enclenche le handler | OK |

# Compteur

## Analyse

Pour le compteur rien d’énorme n’a été fait du côté C. Il nous fallait avoir la possibilité d’enable/disable le compteur, du reste, ainsi que de récupérer la valeur actuelle, cela ne nécessite qu’une logique de lecture et d’écriture des registres et ne sera donc pas expliqué en détail dans la conception

## Conception

void enable\_counter(){

\*(ENABLE\_COUNTER\_REG) = ENABLE\_COUNTER\_MASK;

}

void disable\_counter(){

\*(ENABLE\_COUNTER\_REG) = 0;

}

uint32\_t counter\_current\_value(){

return \*(COUNTER\_CURRENT\_VALUE\_REG);

}

void reset\_counter(){

\*RESET\_COUNTER\_REG = RESET\_COUNTER\_MASK;

}

## Test

|  |  |
| --- | --- |
| Description | OK/KO |
| Le reset du compteur mais bel et bien le compteur à 0 (compteur désactivé) | OK |
| L'activation du compteur démarre bel et bien notre compteur | OK |
| La désactivation du compteur stop le compteur et ne le reset pas | OK |
| La lecture du compteur lit bien le bon registre et une valeur valide | OK |

# Nombre aléatoire

## Analyse

Pour la génération de nombre aléatoire je me suis servi du fichier /dev/urandom, ce dernier contient des valeurs aléatoires générées à partir de sources d'entropie du système, comme les mouvements de la souris, les frappes clavier, ou les fluctuations matérielles.

## Conception

Pour faire fonctionner cela il nous a fallu ouvrir le fichier /dev/urandom, puis lire sa valeur. Il nous a été demandé que la valeur random dois être entre 1s et 4s. Pour cela nous avons cela = ((random\_value % 4) + 1)

uint32\_t generate\_random(void){

uint32\_t random\_value;

int fd = open("/dev/urandom", O\_RDONLY);

if (fd == -1) {

perror("Erreur ouverture /dev/urandom");

exit(EXIT\_FAILURE);

}

if (read(fd, &random\_value, sizeof(random\_value)) != sizeof(random\_value)) {

perror("Erreur lecture /dev/urandom");

close(fd);

exit(EXIT\_FAILURE);

}

close(fd);

return ((random\_value % 4) + 1);

}

## Test

|  |  |
| --- | --- |
| Description | OK/KO |
| Le fichier /dev/urandom s’ouvre sans erreur | OK |
| Le fichier /dev/urandom lis une valeur valide | OK |
| La valeur lue change à chaque fois et n’a pas de pattern défini (peu importe si le programme et relancée ou/et que la carte et remise à 0) | OK |
| La valeur retournée de le la fonction se situe bien entre 1 et 4 | OK |

# Application principal

## Analyse

Pour l’application principal il n’y a pas eu de grande analyse à faire puisque le fonctionnement principal a été décrit dans le cahier des charges.

## Conception

Les points principaux ce situe dans la boucle while. La première instruction appelle une fonction venant tout simplement afficher la valeur souhaitée en fonction des switches comme cela :

* SW0=1 : affichage du meilleur temps de réaction en ms.
* SW1=1 : affichage du plus mauvais temps de réaction en ms.
* SW2=1 : affichage du nombre d’erreur lors de la mesure du temps de réaction (erreur = appuie sur KEY0 avant le symbole de début).
* SW3=1 : affichage du nombre total de tentative de mesure du temps de réaction.
* Sinon : affichage du dernier temps de réaction obtenue en ms.

No\_error est juste un appel de fonction venant reset notre flag d’erreur.

Une fois key 1 pressé un message est affiché dans la console pour dire que la mesure a démarré, puis on vient afficher sur notre Max10, notre dessin d’attente.

Notre valeur aléatoire d’attente est ensuite générée et le compteur peut démarrer. La seconde while laisse notre programme en attente tant que l’on n'a pas atteint notre valeur aléatoire.

Dès que l’attente et terminée et que key0 n’a pas été pressé avant de finir la while (new\_error contrôle cela) nous allons afficher le symbole de début et redémarrer le compteur de temps.

while(true){

Seg7\_display(Switchs\_read());

if(Key\_read\_edge(1)){

no\_error();

send\_to\_uart("Reaction time measurement started.\r\nWait for the start symbol on Max10\_leds and press KEY0 as fast as possible.\r\n");

//symbole d'attente

Max10\_write\_square(WAIT\_DISPLAY);

new\_attemps();

rand\_value = generate\_random();

reset\_counter();

enable\_counter();

while((counter\_current\_value() \* 0.000000020 <= rand\_value) && !new\_error());

if(!new\_error()){

disable\_counter();

start\_game();

//symbole de début

Max10\_write\_square(BEGIN\_DISPLAY);

reset\_counter();

enable\_counter();

}

}

}

## Test

|  |  |
| --- | --- |
| Description | OK/KO |
| Les constantes affichées son correct | OK |
| L'initialisation de tout le matériel est faite correctement (GIC, Max10, De1-SoC, etc...) | OK |
| L’envoie du message d’accueil se fait correctement | OK |
| Les afficheurs 7 segments affiche bien la valeur correspondante aux switch choisi (pire score, meilleurs scores, etc.) | OK |
| La pression sur key1 (et uniquement key1) démarre l’attente avant la mesure du temps de réaction. | OK |
| La pression multiple de key1 (sans pression de key0) redémarre à chaque fois la mesure | OK |
| La pression de key1 affiche un “+” sur le carré de LED de la Max10 | OK |
| L’attente avant la mesure du temps de réaction dure bien entre 1 et 4 secondes | OK |
| Après l’attente le carré de LED nous affiche bel et bien un grand carré | OK |
| Après l’attente la mesure du temps de réaction et bel et bien lancé | OK |
| La pression sur key1 annule la mesure et recommence une nouvelle attente | OK |
| Lorsque le grand carré est affiché, une pression sur key0 affiche le résultat obtenu (si tous les switches sont à 0) sur les 7 segment et dans la console | OK |
| Une pression sur key0 lorsque la mesure n’est pas lancée ou qu’il est en attente de l’affichage du grand carré, cela nous provoque une erreur et affiche un message dans la console. | OK |
| Une pression sur key0 toggle la LED 9 | OK |

# Emetteur Série UART

## Question

Pourrait-on utiliser un module UART du HPS pour piloter la liaison série asynchrone 20 bits vers la carte Max10\_leds ? Justifier votre réponse.

## Réponse

Non cela ne serait pas possible, car le module UART ne permet d’envoyer que des trames de taille 5,6,7 et 8 bits. Malheureusement ne pouvant pas envoyer soit 20 bits soit bit par bit, le seul moyen serait de configurer la Max10 afin qui puisse reconstituer les différentes trames qu’il reçoit, mais ne sachant pas si la carte Max10 est reconfigurable ou non nous avons considéré que cela n’était pas possible.

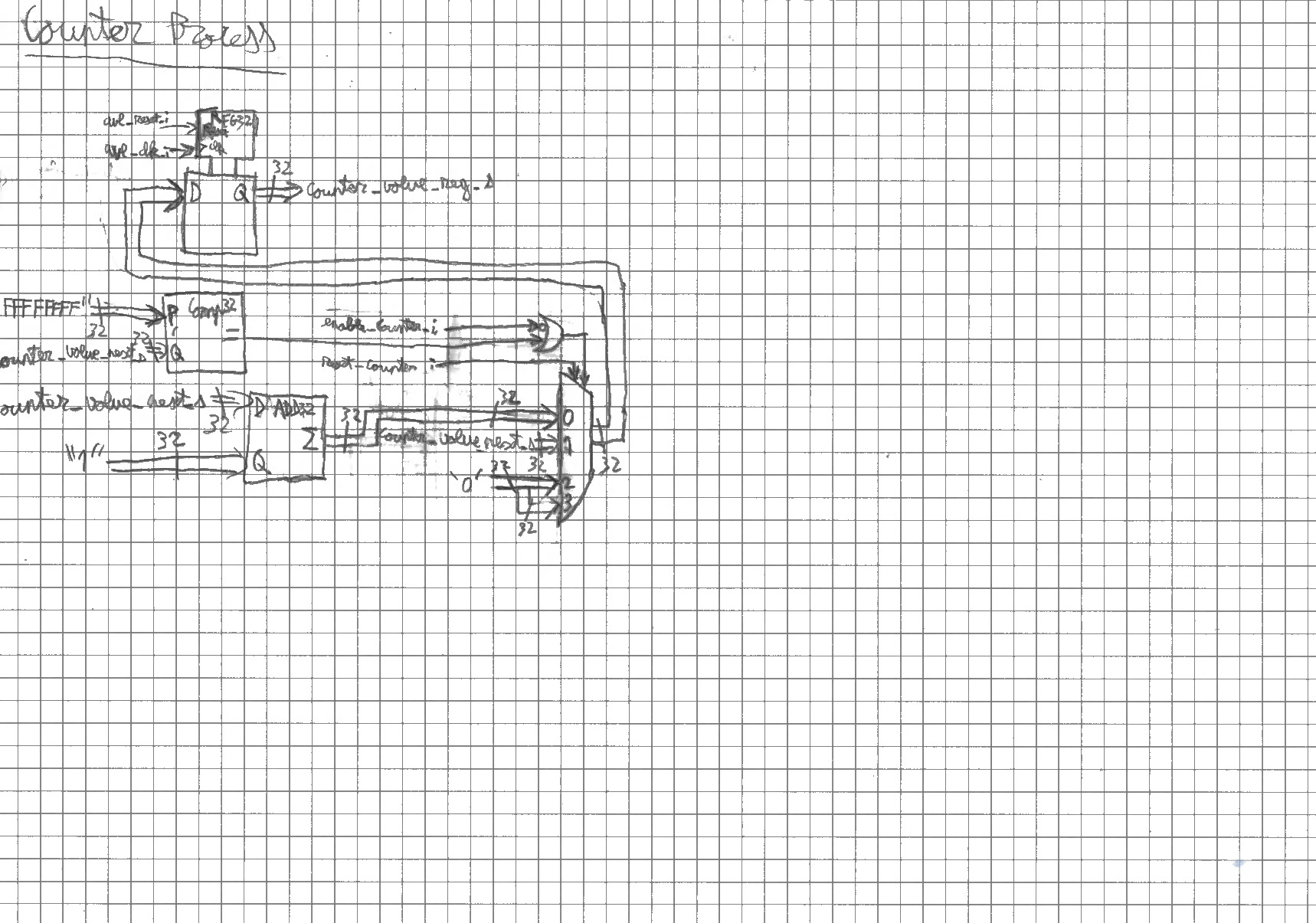
# Partie VHDL

Voici notre plan d’adressage :

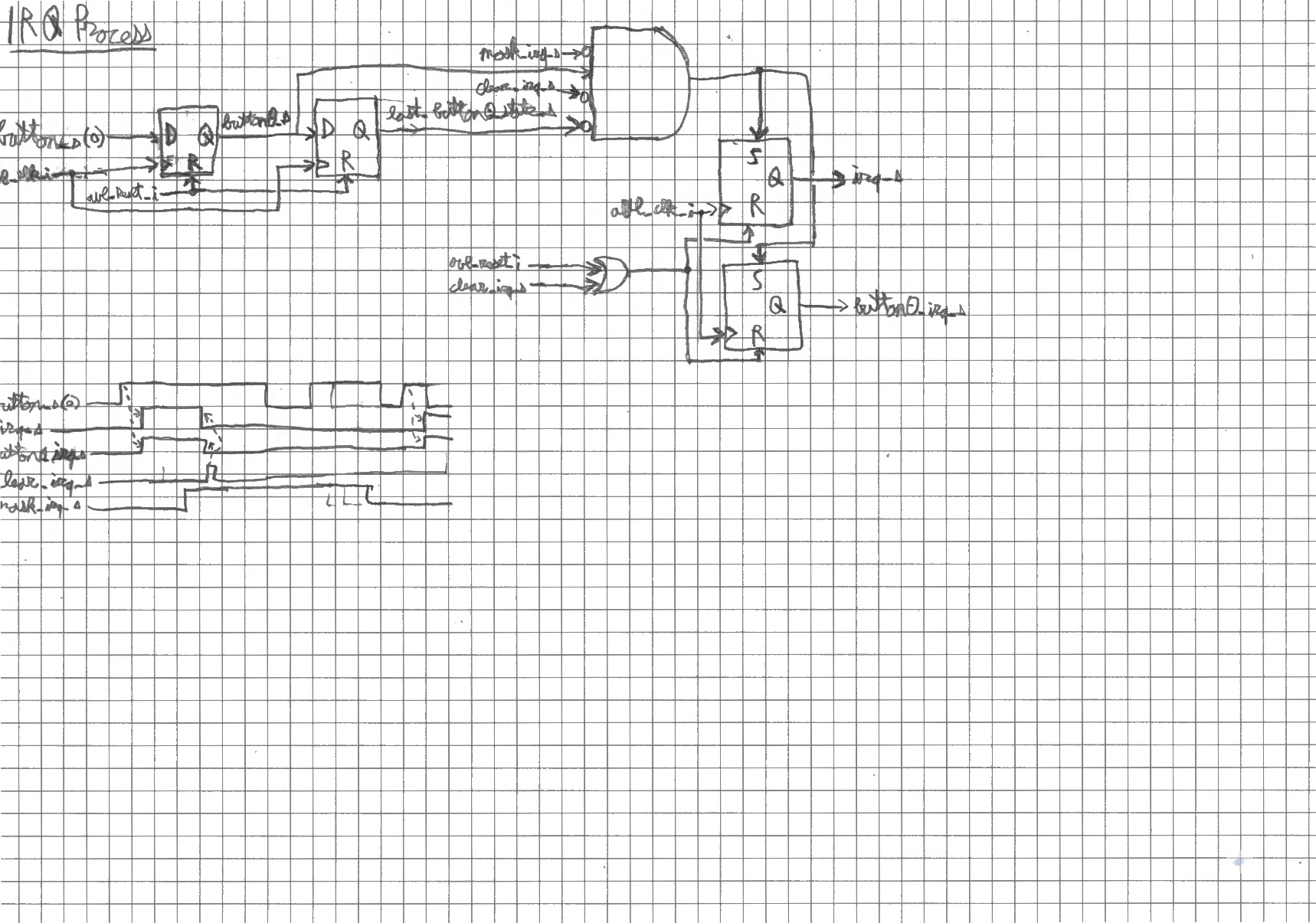
|  |  |  |
| --- | --- | --- |
| Adresse | Rd | Wr |
| 0x00 | [31..0] Int Const ID | not used |
| 0x04 | [31..4] 0x0 & [3..0] buttons | not used |
| 0x08 | [31..10] 0x0 & [9..0] switchs | not used |
| 0x0C | [31..10] 0x0 & [9..0] leds | [31..10] reserved & [9..0] leds |
| 0x10 | [31..28] 0x0 & [27..0] hex3 à hex0 | [31..28] reserved & [27..0] hex3 à hex0 |
| 0x14 *– serial transmitter* | [31..3] 0x0 & [2] f\_ready & [1..0] con\_80p\_status | [31..20] reserved & [19..16] code & [15..0] data |
| 0x18 *– internal counter* | [31..0] counter current value | [31..2] reserved & [1] enable counter & [0] reset counter |
| 0x1C *– IRQ managment* | [31..1] 0x0 & [0] IRQ\_flag | [31..2] 0x0 & [1] mask\_IRQ & [0] clear\_IRQ |

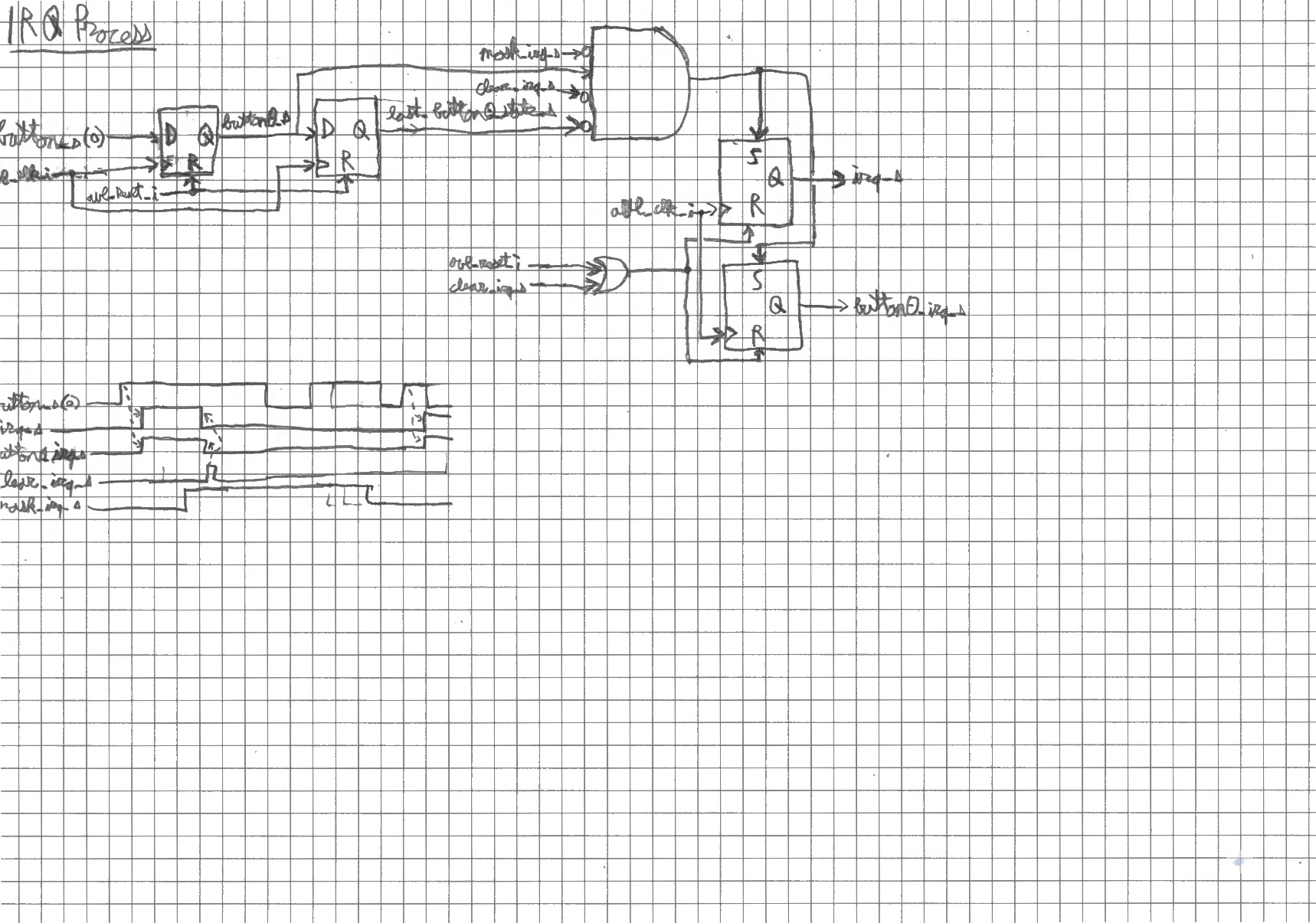
NB : si la valeur du compteur est 3, cela signifie 3\*20ns (donc 60ns et non 3ns)

## Schémas blocs



Le compteur correspond à un simple compteur donc quand l’enable est actif il compte sinon il maintient. Il s’arrête s’il atteint sa valeur max pour ne pas refaire un tour (max+1 = 0). Le reset du compteur est automatiquement désactivé par l’interface (quand on n'écrit pas de l’activer).





button\_s(0)

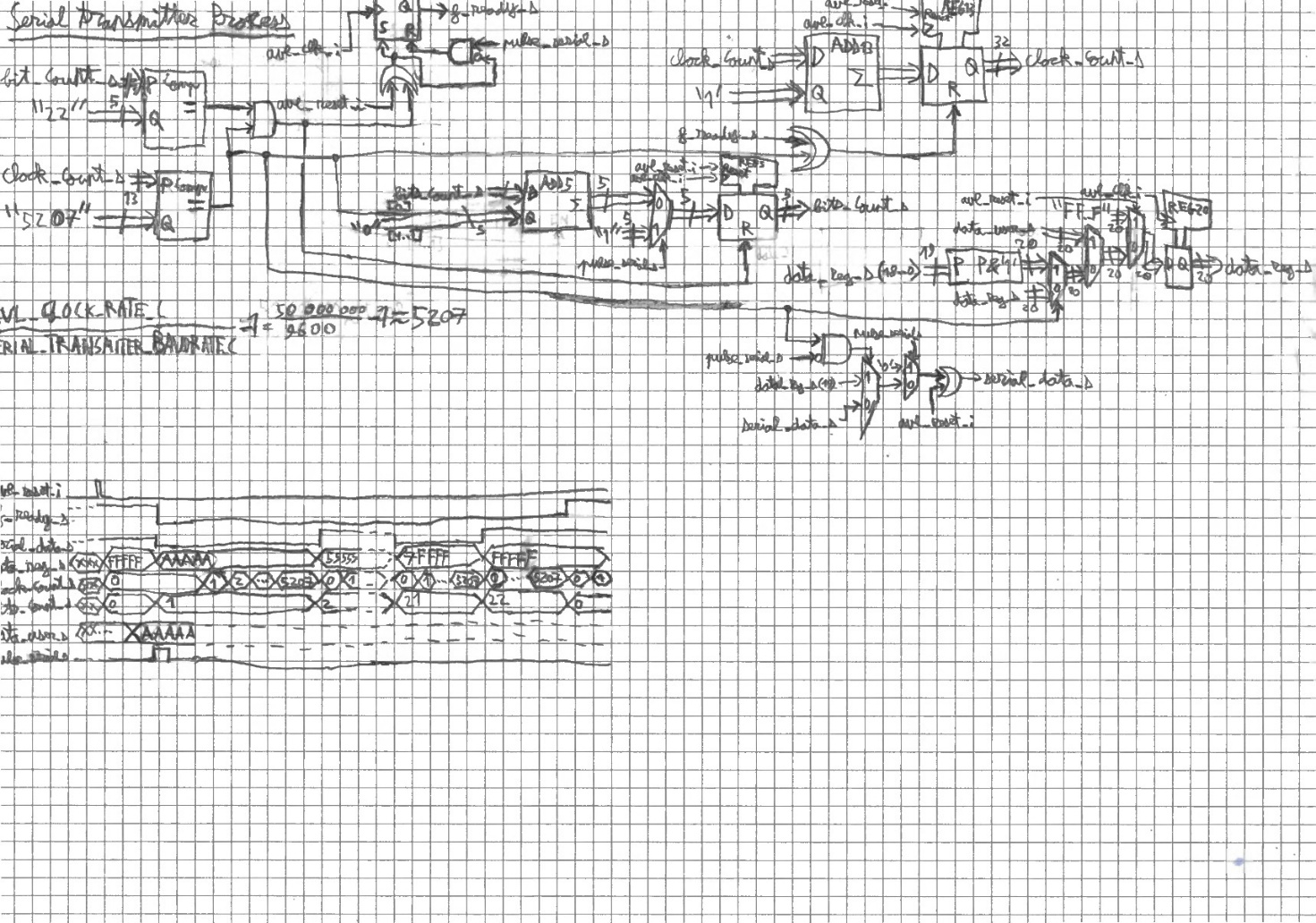
irq\_s

button0\_irq\_s

clear\_irq\_s

mask\_irq\_s

Comme on peut le voir sur le chronogramme button\_irq et irq sont finalement les mêmes signaux. J’ai gardé la distinction entre les deux car s’il on voulait ajouter une irq (quand le compteur à atteint la valeur max par exemple) irq ne serai alors plus le même signale que button\_irq. Dans le schéma j’ai modélisé irq par une filp flop mais ça pourrait être une porte OU des autre flags d’irq (ici button\_irq). Clear\_irq est automatiquement désactivé (à la manière du reset du compteur).



avl\_reset\_i

f\_ready\_s

serial\_data\_s

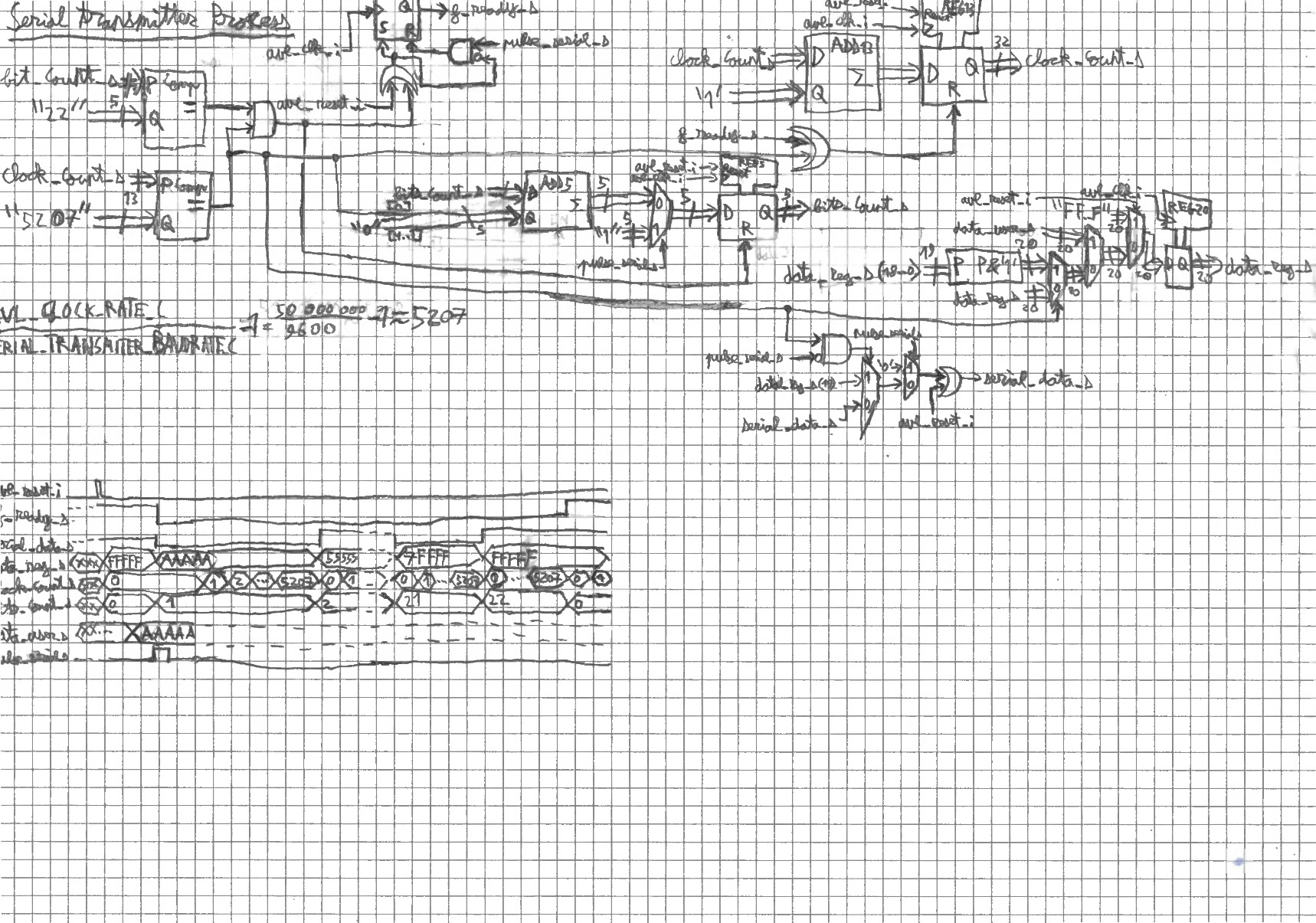
data\_reg\_s

clock\_count\_s

bits\_count\_s

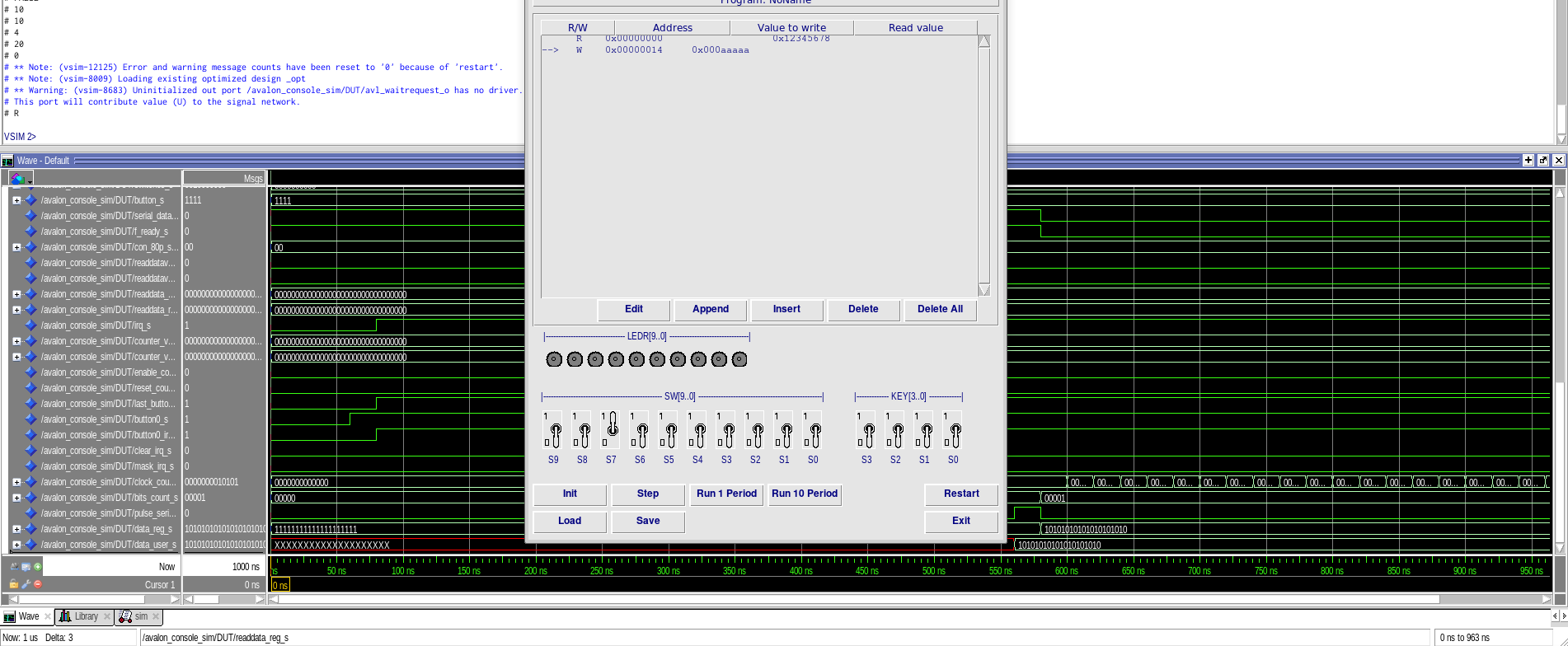
data\_user\_s

pulse\_serial\_s



Pour communiquer avec la ligne sérielle, nous n’avons pas fait de MSS et nous avons plutôt opté pour l’utilisation d’un registre à décalage et de quelques logiques supplémentaires (comparateur, additionneur, …) comme l’illustre le schéma bloc ci-dessus. Pulse\_serial s’active automatiquement lorsqu’on écrit une valeur dans code/data (et se désactive de manière similaire à clear\_irq ou reset\_counter). On ajoute toujours des 1 dans notre décalage pour qu’il écrive le stop bit (qui est à 1) automatiquement.

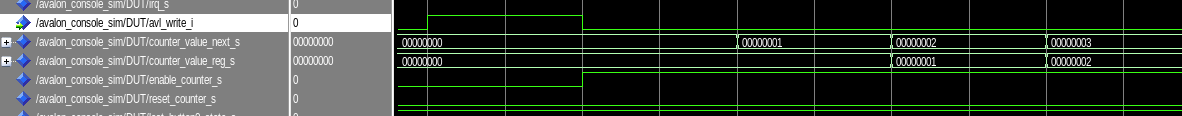
## Simulation



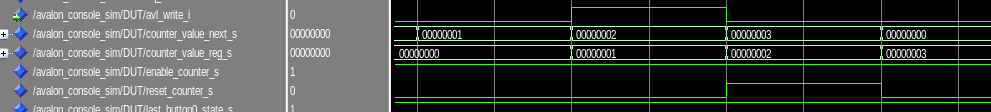
Pour la simulation on a testé avec le script "run\_avalon\_sim.tcl" qui donne une interface similaire à ce qu’il y sur l’image en dessus. Il faut commencer par appuyer sur "Append" et sélectionner une commande (read/write) et dans le cas d’une écriture la valeur à écrire. Ici on test de lire la constante puis d’activer la ligne sérielle avec que des 0xA (1010). Puis appuyer sur "init" et "Step" permet de se trouver dans la première commande à chaque "step" on passe à la commande suivante et Run 1 Period ou Run 10 Period permets de faire passer le temps.

Je n’ai pas retesté ce qui fonctionnait sur l’interface qu’on a récupéré des anciens labos (sauf la lecture de la constante en 0x0). C’est-à-dire les leds, switchs, buttons et affichage hex.

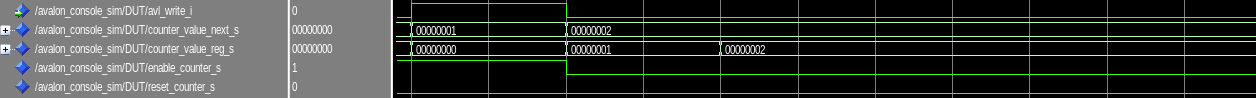
Pour le compteur, j’ai d’abord écrit dans l’enable :

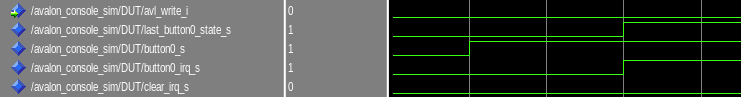


Puis j’ai demandé un reset (du compteur pas de l’interface) :

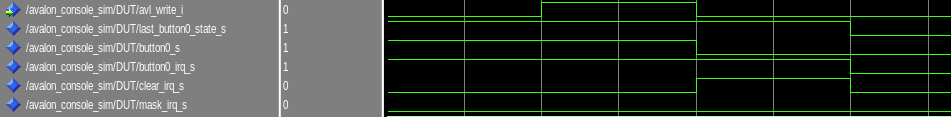


Puis enfin, j’ai désactivé le compteur et j’ai vérifié le maintien :

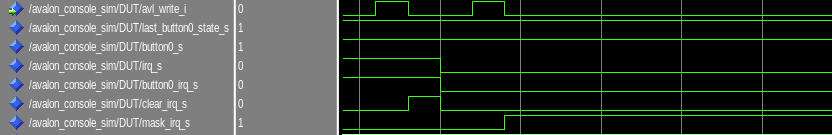


Pour la gestion des irq, j’ai testé le cas standard d’un appui du bouton 0 :  


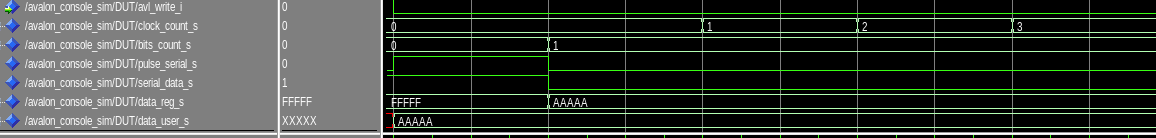
Puis le clear irq :



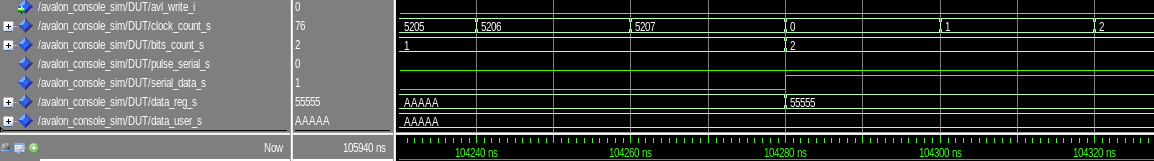
Et finalement le masking de l’IRQ :



Pour le serial transmitter, j’ai simplement vérifié le baudrate (5208 coups de clocks avant de changer) et la désactivation du f\_ready. Je n’ai pas testé le fonctionnement complet car pour attendre la fin d’une transmission il faudrait attendre 114’576 (=5208\*22) coups de clocks ce qui demande d’appuyer 11'458 sur le bouton d’avancée "rapide" de 10 coups de clock :



Début du start bit sur l’image précédente et le passage au MSB du code sur la seconde :



NB : tous les chronogrammes on été fait avec la version 2.0 de l’interface donc avec la détection de flanc du bouton inversée.

# Conclusion

Malgré la longueur de ce labo cela nous a permis de mettre en pratique quasiment tout ce que l’on a pu voir durant le cours d’ARE ainsi que durant les anciens labos. Le plus gros challenge durant ce labo fût de résoudre les problèmes dans le fonctionnement de notre programme. Bien que l’usage du debugger nous aies aidé, la grande quantité de code ainsi que sa modularité (vhdl-c donc parfois on croit que le problème c’est le c alors que c’est le vhdl et inversement), il devient parfois laborieux de trouver la source du problème. Nous avons tout de même réussi à résoudre tous nos problèmes dans les temps pour avoir un programme fonctionnel.

|  |  |
| --- | --- |
| **Date** | 18.12.2024 |
| **Signature de l’étudiant 1** | Guillaume Gonin |
| **Signature de l’étudiant 2** | Patrick Maillard |

# Annexes

**address\_map\_arm.h :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : address\_map\_arm.h

\* Author : Anthony Convers

\* Date : 27.10.2022

\*

\* Context : ARE lab

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: provides address values that exist in the ARM system

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Engineer Comments

\* 0.0 27.10.2022 ACS Initial version.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define BOARD "DE1-SoC"

/\* Memory \*/

#define DDR\_BASE 0x00000000

#define DDR\_END 0x3FFFFFFF

#define A9\_ONCHIP\_BASE 0xFFFF0000

#define A9\_ONCHIP\_END 0xFFFFFFFF

#define SDRAM\_BASE 0xC0000000

#define SDRAM\_END 0xC3FFFFFF

#define FPGA\_ONCHIP\_BASE 0xC8000000

#define FPGA\_ONCHIP\_END 0xC803FFFF

#define FPGA\_CHAR\_BASE 0xC9000000

#define FPGA\_CHAR\_END 0xC9001FFF

**avalon\_function.c :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : avalon\_function.h

\* Author : Rafael Dousse & Patrick Maillard

\* Date : 09.11.2024

\*

\* Context : ARE lab

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: Avalon functions

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Student Comments

\* 0.0 09.11.2024 RAD & PAM Initial version.

\* 1.0 26.01.25 MaillardP modified version for ARE lab6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "avalon\_function.h"

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

const unsigned char seven\_segment\_map[16] = {

0b1000000, // 0

0b1111001, // 1

0b0100100, // 2

0b0110000, // 3

0b0011001, // 4

0b0010010, // 5

0b0000010, // 6

0b1111000, // 7

0b0000000, // 8

0b0010000, // 9

0b0001000, // A

0b0000011, // b

0b1000110, // C

0b0100001, // d

0b0000110, // E

0b0001110 // F

};

// All the value for our time measurement

float my\_reaction\_time = 0;

float best\_reaction\_time = 9.999;

float worst\_reaction\_time = 0;

uint32\_t nbr\_of\_error = 0;

uint32\_t nbr\_of\_attempt = 0;

// flag

bool game\_started = false;

bool error = false;

uint32\_t generate\_random(void){

uint32\_t random\_value;

int fd = open("/dev/urandom", O\_RDONLY);

if (fd == -1) {

perror("Erreur ouverture /dev/urandom");

exit(EXIT\_FAILURE);

}

if (read(fd, &random\_value, sizeof(random\_value)) != sizeof(random\_value)) {

perror("Erreur lecture /dev/urandom");

close(fd);

exit(EXIT\_FAILURE);

}

close(fd);

return ((random\_value % 4) + 1);

}

uint32\_t Switchs\_read(void)

{

volatile uint32\_t \*value\_switchs = SWITCH\_REG;

return ((\*value\_switchs & SWITCHS\_BITS));

}

void Segs7\_init(void){

Seg7\_write\_hex(0, 0);

Seg7\_write\_hex(1, 0);

Seg7\_write\_hex(2, 0);

Seg7\_write\_hex(3, 0);

}

void Seg7\_write(int seg7\_number, uint32\_t value){

uint32\_t hex\_mask;

switch(seg7\_number){

case 0:

hex\_mask = HEX0\_BITS;

break;

case 1:

hex\_mask = HEX1\_BITS;

value = value << 7;

break;

case 2:

hex\_mask = HEX2\_BITS;

value = value << 14;

break;

case 3:

hex\_mask = HEX3\_BITS;

value = value << 21;

break;

default:

printf("7-segment chosen does not exist or shouldn't be used");

return;

}

volatile uint32\_t \*value\_hex = INTERFACE\_REG(HEX\_OFFSET);

\*value\_hex &= ~hex\_mask;

\*value\_hex |= (~value & hex\_mask);

}

void Seg7\_reaction\_time(double reaction\_time){

uint32\_t hex0\_to\_3;

uint8\_t hex0, hex1, hex2, hex3;

int mynumber = (double)reaction\_time \* 1000;

if (mynumber > 9999) {

mynumber = 9999;

}

hex0 = mynumber % 10;

hex1 = (mynumber / 10) % 10;

hex2 = (mynumber / 100) % 10;

hex3 = (mynumber / 1000) % 10;

hex0\_to\_3 = (seven\_segment\_map[hex3] << 21) | (seven\_segment\_map[hex2] << 14) | (seven\_segment\_map[hex1] << 7) | seven\_segment\_map[hex0];

volatile uint32\_t \*value\_hex = INTERFACE\_REG(HEX\_OFFSET);

\*value\_hex &= 0xFFFFFFFF;

\*value\_hex = (hex0\_to\_3);

}

void Seg7\_write\_all(uint32\_t number){

uint32\_t hex0\_to\_3;

uint8\_t hex0, hex1, hex2, hex3;

if (number > 9999) {

number = 9999;

}

hex0 = number % 10;

hex1 = (number / 10) % 10;

hex2 = (number / 100) % 10;

hex3 = (number / 1000) % 10;

hex0\_to\_3 = (seven\_segment\_map[hex3] << 21) | (seven\_segment\_map[hex2] << 14) | (seven\_segment\_map[hex1] << 7) | seven\_segment\_map[hex0];

volatile uint32\_t \*value\_hex = INTERFACE\_REG(HEX\_OFFSET);

\*value\_hex &= 0xFFFFFFFF;

\*value\_hex = (hex0\_to\_3);

}

void Seg7\_display(uint32\_t switch\_value){

if(switch\_value & SWITCH0\_MASK){

Seg7\_reaction\_time(best\_reaction\_time);

}

else if(switch\_value & SWITCH1\_MASK){

Seg7\_reaction\_time(worst\_reaction\_time);

}

else if(switch\_value & SWITCH2\_MASK){

Seg7\_write\_all(nbr\_of\_error);

}

else if(switch\_value & SWITCH3\_MASK){

Seg7\_write\_all(nbr\_of\_attempt);

}

else{

Seg7\_reaction\_time(my\_reaction\_time);

}

}

void Seg7\_write\_hex(int seg7\_number, uint32\_t value){

if (value > 0xF){

printf("Value chosen isn't an hexadecimal value");

return;

}

uint32\_t hex\_mask;

uint32\_t shift = 0;

switch(seg7\_number){

case 0:

hex\_mask = HEX0\_BITS;

break;

case 1:

hex\_mask = HEX1\_BITS;

shift = 7;

break;

case 2:

hex\_mask = HEX2\_BITS;

shift = 14;

break;

case 3:

hex\_mask = HEX3\_BITS;

shift= 21;

break;

default:

printf("7-segment chosen does not exist or shouldn't be used");

return;

}

volatile uint32\_t \*value\_hex = INTERFACE\_REG(HEX\_OFFSET);

\*value\_hex &= ~hex\_mask;

\*value\_hex |= (seven\_segment\_map[value] << shift) & hex\_mask;

}

void Leds\_write(uint32\_t value)

{

if (value > 0x3FF)

{

printf("Error: value is too big\n");

return;

}

volatile uint32\_t \*value\_leds = LED\_REG;

\*value\_leds = value; // Remplace directement la valeur des LEDs

}

void Leds\_set(uint32\_t maskleds)

{

if (maskleds > 0x3ff)

{

printf("Error: maskleds is too big\n");

return;

}

volatile uint32\_t \*value\_leds = LED\_REG;

\*value\_leds |= maskleds; // Allume les LEDs spécifiées par le masque

}

uint32\_t Leds\_read(uint32\_t led)

{

volatile uint32\_t \*value\_leds = LED\_REG;

return (\*value\_leds & led);

}

void Leds\_toggle(uint32\_t maskleds){

volatile uint32\_t \*value\_leds = LED\_REG;

\*value\_leds ^= maskleds & LEDS\_BITS;

}

bool Key\_read(int key\_number)

{

volatile uint32\_t \*dataValue = BOUTON\_REG;

uint32\_t my\_key\_value = (\*dataValue >> key\_number) & 0x1;

return !my\_key\_value;

}

// Variable to store the state of the keys

bool positive\_edge[4] = {false, false, false, false};

bool Key\_read\_edge(int key\_number)

{

volatile uint32\_t \*dataValue = BOUTON\_REG;

// Get the key number and check its value

uint32\_t my\_key\_value = (\*dataValue >> key\_number) & 0x1;

if (my\_key\_value == 0 && !positive\_edge[key\_number])

{ // Key is pressed (active low) and no edge detected yet

positive\_edge[key\_number] = true; // Mark the positive edge (key press) as detected

return true; // Rising edge (key press) detected

}

else if (my\_key\_value == 1 && positive\_edge[key\_number])

{ // Key is released (active low)

positive\_edge[key\_number] = false; // Reset the edge detection for the next press

}

return false;

}

void Max10\_init(void){

uint32\_t code = 0x0;

uint32\_t value = 0x0;

volatile uint32\_t \*max10\_data\_reg = SERIAL\_LINK\_DATA\_REG;

// loop over all the possible code setted

while(code <= 0xA){

if(serial\_transmitter\_ready()){

uint32\_t myserial = code << SERIAL\_LINK\_CODE\_SHIFT | value << SERIAL\_LINK\_DATA\_SHIFT;

\*max10\_data\_reg = myserial;

code++;

}

}

uint32\_t myserial = code << SERIAL\_LINK\_CODE\_SHIFT | value << SERIAL\_LINK\_DATA\_SHIFT;

\*max10\_data\_reg = myserial;

}

bool Max10\_check\_status(void)

{

volatile uint32\_t \*status = STATUS\_REG;

return ((\*status & STATUS\_VALID) == STATUS\_VALID);

}

void Max10\_write\_serial\_link(uint32\_t value, uint8\_t code)

{

volatile uint32\_t \*max10\_data\_reg = SERIAL\_LINK\_DATA\_REG;

if (code > 0x0A)

{

printf("Erreur : Valeur du code invalide.\n");

return;

}

// Wait till the MAX10 tell us we can send another data

while(!serial\_transmitter\_ready()){}

uint32\_t myserial = code << SERIAL\_LINK\_CODE\_SHIFT | value << SERIAL\_LINK\_DATA\_SHIFT;

\*max10\_data\_reg = myserial;

}

void Max10\_write\_square(uint32\_t value){

uint32\_t value\_MSB = value >> 16;

uint32\_t value\_LSB = value & 0xFFFF;

Max10\_write\_serial\_link(value\_MSB, 0x7);

Max10\_write\_serial\_link(value\_LSB, 0x6);

}

bool serial\_transmitter\_ready(){

volatile uint32\_t \*ready = SERIAL\_LINK\_READY\_REG;

uint32\_t reg\_val = \*ready;

return ((reg\_val & SERIAL\_LINK\_READY\_MASK) == SERIAL\_LINK\_READY\_MASK);

}

void enable\_counter(){

\*(ENABLE\_COUNTER\_REG) = ENABLE\_COUNTER\_MASK;

}

void disable\_counter(){

\*(ENABLE\_COUNTER\_REG) = 0;

}

uint32\_t counter\_current\_value(){

return \*(COUNTER\_CURRENT\_VALUE\_REG);

}

void reset\_counter(){

\*RESET\_COUNTER\_REG = RESET\_COUNTER\_MASK;

}

void clear\_irq(){

volatile uint32\_t \*clear = CLEAR\_IRQ\_REG;

\*clear |= IRQ\_CLEAR\_MASK;

}

void new\_time(){

// if key0 was pressed when the measurement began

if(game\_started){

// so set all the reaction time

my\_reaction\_time = (float)counter\_current\_value() \* (float)CLOCK\_IN\_SECOND;

if(best\_reaction\_time > my\_reaction\_time){

best\_reaction\_time = my\_reaction\_time;

}

if(worst\_reaction\_time < my\_reaction\_time){

worst\_reaction\_time = my\_reaction\_time;

}

// Ending symbol

Max10\_write\_square(END\_DISPLAY);

}

else{

// if the key0 was pressed before the measurement began we have an error

send\_to\_uart("Error: Premature key press detected.\r\nMeasurement aborted.\r\n");

nbr\_of\_error++;

error = true;

// display off

Max10\_write\_square(0x0);

}

char buffer[256]; // Create a buffer to hold the formatted string

sprintf(buffer,

"Reaction time: %f s\r\n"

"Best time: %f s\r\n"

"Worst time: %f s\r\n"

"Errors: %d\r\n"

"Total attempts: %d\r\n",

my\_reaction\_time, best\_reaction\_time, worst\_reaction\_time, nbr\_of\_error, nbr\_of\_attempt);

send\_to\_uart(buffer);

disable\_counter();

game\_started = false;

}

void start\_game(){

game\_started = true;

}

void new\_attemps(){

nbr\_of\_attempt++;

}

bool new\_error(){

return error;

}

void no\_error(){

error = false;

}

**avalon\_function.h :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : avalon\_function.h

\* Author : Rafael Dousse & Patrick Maillard

\* Date : 09.11.2024

\*

\* Context : ARE lab

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: Header file for avalon function

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Student Comments

\* 0.0 09.11.2024 RAD & PAM Initial version.

\* 1.0 26.01.25 MaillardP modified version for ARE lab6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdbool.h>

#include <stdint.h>

#include <stdio.h>

#include "axi\_lw.h"

// Base address

#define H2F\_BASE\_ADD AXI\_LW\_HPS\_FPGA\_BASE\_ADD

#define INTERFACE\_BASE\_ADD 0xFF210000

// ACCESS MACROS

#define INTERFACE\_REG(\_x\_) ((volatile uint32\_t \*)(INTERFACE\_BASE\_ADD + \_x\_)) // \_x\_ is an offset with respect to the base address

//#define H2F(\_x\_) (volatile uint32\_t \*)(H2F\_BASE\_ADD + \_x\_)

// Offset interface

#define ID\_OFFSET 0x0

#define LED\_OFFSET 0xc

#define SWITCH\_OFFSET 0x8

#define BOUTON\_OFFSET 0x4

#define HEX\_OFFSET 0x10

// Our added adress

#define CON\_80P\_STATUS\_OFFSET 0x14

#define SERIAL\_LINK\_READY\_OFFSET 0x14

#define SERIAL\_LINK\_CODE\_OFFSET 0x14

#define SERIAL\_LINK\_DATA\_OFFSET 0x14

#define COUNTER\_CURRENT\_VALUE\_OFFSET 0x18

#define ENABLE\_COUNTER\_OFFSET 0x18

#define RESET\_COUNTER\_OFFSET 0x18

#define IRQ\_FLAG\_OFFSET 0x1c

#define CLEAR\_IRQ\_OFFSET 0x1c

// HEX mask

#define HEX3\_BITS 0x0FE00000

#define HEX2\_BITS 0x001FC000

#define HEX1\_BITS 0x00003F80

#define HEX0\_BITS 0x0000007F

// REGISTER

#define ID\_REG INTERFACE\_REG(ID\_OFFSET)

#define LED\_REG INTERFACE\_REG(LED\_OFFSET)

#define SWITCH\_REG INTERFACE\_REG(SWITCH\_OFFSET)

#define BOUTON\_REG INTERFACE\_REG(BOUTON\_OFFSET)

#define STATUS\_REG INTERFACE\_REG(CON\_80P\_STATUS\_OFFSET)

#define SERIAL\_LINK\_CODE\_REG INTERFACE\_REG(CON\_80P\_STATUS\_OFFSET)

#define SERIAL\_LINK\_READY\_REG INTERFACE\_REG(SERIAL\_LINK\_READY\_OFFSET)

#define SERIAL\_LINK\_DATA\_REG INTERFACE\_REG(SERIAL\_LINK\_CODE\_OFFSET)

#define COUNTER\_CURRENT\_VALUE\_REG INTERFACE\_REG(COUNTER\_CURRENT\_VALUE\_OFFSET)

#define ENABLE\_COUNTER\_REG INTERFACE\_REG(ENABLE\_COUNTER\_OFFSET)

#define RESET\_COUNTER\_REG INTERFACE\_REG(RESET\_COUNTER\_OFFSET)

#define IRQ\_FLAG\_REG INTERFACE\_REG(IRQ\_FLAG\_OFFSET)

#define CLEAR\_IRQ\_REG INTERFACE\_REG(CLEAR\_IRQ\_OFFSET)

// Button mask

#define BOUTON\_BITS 0x0000000F

// Serial link mask

#define SERIAL\_LINK\_CODE\_MASK 0x000F0000

#define SERIAL\_LINK\_DATA\_MASK 0x0000FFFF

#define SERIAL\_LINK\_READY\_MASK 0x00000004

// Counter mask

#define ENABLE\_COUNTER\_MASK 0x00000002

#define RESET\_COUNTER\_MASK 0x00000001

// IRQ mask

#define IRQ\_CLEAR\_MASK 0x00000001

#define IRQ\_MASK\_MASK 0x00000002

// Switch mask

#define SWITCHS\_BITS 0x000003FF

#define SWITCH0\_MASK 0x00000001

#define SWITCH1\_MASK 0x00000002

#define SWITCH2\_MASK 0x00000004

#define SWITCH3\_MASK 0x00000008

// Shift value

#define SERIAL\_LINK\_CODE\_SHIFT 16

#define SERIAL\_LINK\_DATA\_SHIFT 0

#define ENABLE\_COUNTER\_SHIFT 1

#define RESET\_COUNTER\_SHIFT 0

// Led mask

#define LEDS\_BITS 0x000003FF

#define LED0 (uint32\_t)(1 << 0)

#define LED1 (uint32\_t)(1 << 1)

#define LED2 (uint32\_t)(1 << 2)

#define LED3 (uint32\_t)(1 << 3)

#define LED4 (uint32\_t)(1 << 4)

#define LED5 (uint32\_t)(1 << 5)

#define LED6 (uint32\_t)(1 << 6)

#define LED7 (uint32\_t)(1 << 7)

#define LED8 (uint32\_t)(1 << 8)

#define LED9 (uint32\_t)(1 << 9)

// Display value

#define WAIT\_DISPLAY 0x00847C84

#define BEGIN\_DISPLAY 0x03F1463F

#define END\_DISPLAY 0x000E39C0

// Other value

#define CLOCK\_IN\_SECOND 0.000000020

#define LED\_OFF 0x0

#define NUM\_MODE 4

#define STATUS\_VALID 0x01

// Leds\_write function : Write a value to all Leds (LED9 to LED0)

// Parameter : "value"= data to be applied to all Leds

// Return : None

void Leds\_write(uint32\_t value);

// Leds\_set function : Set to ON some or all Leds (LED9 to LED0)

// Parameter : "maskleds"= Leds selected to apply a set (maximum 0x3FF)

// Return : None

void Leds\_set(uint32\_t maskleds);

// Leds\_read function : Read the value of all Leds (LED9 to LED0)

// Parameter : "led"= mask to select the Leds to read

// Return : Value of the Led

uint32\_t Leds\_read(uint32\_t led);

// Leds\_toggle function : Toggle the curent value of some or all Leds (LED9 to LED0)

// Parameter : "maskleds"= Leds selected to apply a toggle (maximum 0x3FF)

// Return : None

void Leds\_toggle(uint32\_t maskleds);

// Key\_read function : Read one Key status, pressed or not (KEY0 or KEY1 or KEY2 or KEY3)

// Parameter : "key\_number"= select the key number to read, from 0 to 3

// Return : True(1) if key is pressed, and False(0) if key is not pressed

bool Key\_read(int key\_number);

// Key\_read\_edge function : Read one Key status, pressed or not but only on the rising edge

// Parameter : "key\_number"= select the key number to read, from 0 to 3

// Return : True(1) if key is pressed, and False(0) if key is not pressed

bool Key\_read\_edge(int key\_number);

// Switchs\_read function : Read the switchs value

// Parameter : None

// Return : Value of all Switchs (SW9 to SW0)

uint32\_t Switchs\_read(void);

// Max10\_init function : All LED are turned off on the MAX10

// Parameter : None

// Return : None

void Max10\_init(void);

// Max10\_check\_status function : Check the status of the MAX10

// Parameter : None

// Return : True(1) if the status is valid, and False(0) if the status is not valid

bool Max10\_check\_status(void);

// Max10\_write\_serial\_link function : Write a value to the MAX10

// Parameter : "value" = data to be applied to the MAX10

// Parameter : "sel" = select the leds to write

// Return : None

void Max10\_write\_serial\_link(uint32\_t value, uint8\_t sel);

// Max10\_write\_square function : Write a value to the square LEDs of the MAX10

// Parameter : "value" = data to be applied to the square LEDs of the MAX10

// Return : None

void Max10\_write\_square(uint32\_t value);

// Segs7\_init function : Initialize all 7-segments display on the DE1-SoC (HEX3 to HEX0)

// Parameter : "value" = None

// Return : None

void Segs7\_init(void);

// Seg7\_write function : Write digit segment value to one 7-segments display (HEX0 or HEX1 or HEX2 or HEX3)

// Parameter : "seg7\_number"= select the 7-segments number, from 0 to 3

// Parameter : "value"= digit segment value to be applied on the selected 7-segments (maximum 0x7F to switch ON all segments)

// Return : None

void Seg7\_write(int seg7\_number, uint32\_t value);

// Seg7\_write\_hex function : Write an Hexadecimal value to one 7-segments display (HEX0 or HEX1 or HEX2 or HEX3)

// Parameter : "seg7\_number"= select the 7-segments number, from 0 to 3

// Parameter : "value"= Hexadecimal value to be display on the selected 7-segments, form 0x0 to 0xF

// Return : None

void Seg7\_write\_hex(int seg7\_number, uint32\_t value);

// Seg7\_display function : Write the best reaction time, worst reaction time, actual reaction time, number of attempt or number of error based of the switch value (HEX3 to HEX0)

// Parameter : "switch\_value" = value of our switches

// Return : None

void Seg7\_display(uint32\_t switch\_value);

// Seg7\_write\_all function : Write a value to the HEX display (HEX3 to HEX0)

// Parameter : "number" = value to display

// Return : None

void Seg7\_write\_all(uint32\_t number);

// Seg7\_reaction\_time function : Write the value of a reaction time to the HEX display (HEX3 to HEX0)

// Parameter : "reaction\_time" = reaction time to display

// Return : None

void Seg7\_reaction\_time(double reaction\_time);

// enable\_counter function : Start the counter

// Parameter : None

// Return : None

void enable\_counter();

// enable\_counter function : Stop the counter

// Parameter : None

// Return : None

void disable\_counter();

// counter\_current\_value function : Return the current value of our counter

// Parameter : None

// Return : courrent value of the counter

uint32\_t counter\_current\_value();

// enable\_counter function : Reset the counter to 0

// Parameter : None

// Return : None

void reset\_counter();

// clear\_irq function : Clear the IRQ received from our Key0

// Parameter : None

// Return : None

void clear\_irq();

// new\_time function : Called when there is a new measurement.

// It define the current reaction time and also if it is worse or better than all the previous attempt.

// It also display the best reaction time, worst reaction time, actual reaction time, number of attempt and number of error on the console

// Parameter : None

// Return : None

void new\_time();

// new\_attemps function : Increase by 1 the number of attempt

// Parameter : None

// Return : None

void new\_attemps();

// start\_game function : Set the flag that says the measurement has started

// Parameter : None

// Return : None

void start\_game();

// new\_error function : Return if there was an error detected or not

// Parameter : None

// Return : True if there is an error, False if there is none

bool new\_error();

// serial\_transmitter\_ready function : Tell us if the MAX10 is ready to receive a new value or not

// Parameter : None

// Return : True if the MAX10 can send a new value, False if he can't send a new value

bool serial\_transmitter\_ready();

// generate\_random function : Generate a new random value between 1 and 4

// Parameter : None

// Return : Random value between 1 and 4

uint32\_t generate\_random(void);

**axi\_lw.h :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : axi\_lw.h

\* Author : Anthony Convers

\* Date : 27.07.2022

\*

\* Context : ARE lab

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: Header file for bus AXI lightweight HPS to FPGA defines definition

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Student Comments

\* 0.0 27.07.2022 ACS Initial version.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdint.h>

// Base address

#define AXI\_LW\_HPS\_FPGA\_BASE\_ADD 0xFF200000

// ACCESS MACROS

#define AXI\_LW\_REG(\_x\_) \*(volatile uint32\_t \*)(AXI\_LW\_HPS\_FPGA\_BASE\_ADD + \_x\_) // \_x\_ is an offset with respect to the base address

**exceptions.c :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : execptions.c

\* Author : Anthony Convers

\* Date : 27.10.2022

\*

\* Context : Configure the exception/IRQ

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: defines exception vectors for the A9 processor

\* provides code that sets the IRQ mode stack, and that dis/enables interrupts

\* provides code that initializes the generic interrupt controller

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Engineer Comments

\* 0.0 27.10.2022 ACS Initial version.

\* 1.0 26.01.25 MaillardP Finished version

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdint.h>

#include "address\_map\_arm.h"

#include "int\_defines.h"

#include "avalon\_function.h"

/\* This file:

\* 1. defines exception vectors for the A9 processor

\* 2. provides code that sets the IRQ mode stack, and that dis/enables interrupts

\* 3. provides code that initializes the generic interrupt controller

\*/

void fpga\_ISR(void){

new\_time();

Leds\_toggle(LED9);

clear\_irq();

}

// Define the IRQ exception handler

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_irq (void)

{

// Read CPU Interface registers to determine which peripheral has caused an interrupt

int interrupt\_ID = \*((int \*)0xFFFEC10C);

// Handle the interrupt if it comes from the fpga

if (interrupt\_ID == 72) // check if interrupt is from the KEYs (FPGA\_IRQ0)

fpga\_ISR();

else

while (1); // if unexpected, then stay here

// Clear interrupt from the CPU Interface

\*((int \*)0xFFFEC110) = interrupt\_ID;

return;

}

// Define the remaining exception handlers

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_reset (void)

{

while(1);

}

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_undef (void)

{

while(1);

}

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_swi (void)

{

while(1);

}

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_pabort (void)

{

while(1);

}

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_dabort (void)

{

while(1);

}

void \_\_attribute\_\_ ((interrupt)) \_\_cs3\_isr\_fiq (void)

{

while(1);

}

/\*

\* Initialize the banked stack pointer register for IRQ mode

\*/

void set\_A9\_IRQ\_stack(void)

{

uint32\_t stack, mode;

stack = A9\_ONCHIP\_END - 7; // top of A9 onchip memory, aligned to 8 bytes

/\* change processor to IRQ mode with interrupts disabled \*/

mode = INT\_DISABLE | IRQ\_MODE;

asm("msr cpsr, %[ps]" : : [ps] "r" (mode));

/\* set banked stack pointer \*/

asm("mov sp, %[ps]" : : [ps] "r" (stack));

/\* go back to SVC mode before executing subroutine return! \*/

mode = INT\_DISABLE | SVC\_MODE;

asm("msr cpsr, %[ps]" : : [ps] "r" (mode));

}

/\*

\* Turn on interrupts in the ARM processor

\*/

void enable\_A9\_interrupts(void)

{

uint32\_t status = SVC\_MODE | INT\_ENABLE;

asm("msr cpsr, %[ps]" : : [ps]"r"(status));

}

void config\_interrupt(int N, int CPU\_target) {

int reg\_offset, index, value, address;

/\* Configure the Interrupt Set-Enable Registers (ICDISERn).

\* reg\_offset = (integer\_div(N / 32) \* 4

\* value = 1 << (N mod 32) \*/

reg\_offset = (N >> 3) & 0xFFFFFFFC;

index = N & 0x1F;

value = 0x1 << index;

address = 0xFFFED100 + reg\_offset;

/\* Now that we know the register address and value, set the appropriate bit \*/

\*(int \*)address |= value;

/\* Configure the Interrupt Processor Targets Register (ICDIPTRn)

\* reg\_offset = integer\_div(N / 4) \* 4

\* index = N mod 4 \*/

reg\_offset = (N & 0xFFFFFFFC);

index = N & 0x3;

address = 0xFFFED800 + reg\_offset + index;

/\* Now that we know the register address and value, write to (only) the

\* appropriate byte \*/

\*(char \*)address = (char)CPU\_target;

}

/\*

\* Configure the Generic Interrupt Controller (GIC)

\*/

void config\_GIC(void)

{

/\*\*\*\*\*\*\*\*\*\*\*

\* TO DO

\*\*\*\*\*\*\*\*\*\*/

config\_interrupt (72, 1);

// Set Interrupt Priority Mask Register (ICCPMR). Enable interrupts of all

// priorities

\*((int \*) 0xFFFEC104) = 0xFFFF; //0xFF instead of 0xFFFF because of our 8 bit priority value for our ICCPMR at the adress 0xFFFEC104.

// Set CPU Interface Control Register (ICCICR). Enable signaling of

// interrupts

\*((int \*) 0xFFFEC100) = 1;

// Configure the Distributor Control Register (ICDDCR) to send pending

// interrupts to CPUs

\*((int \*) 0xFFFED000) = 1;

}

**hps\_application.c :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : hps\_application.c

\* Author : Maillard Patrick

\* Date : 26.01.25

\*

\* Context : Main program

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: Mesure du temps de reaction avec la carte DE1-SoC et MAX10

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Student Comments

\* 1.0 26.01.25 MaillardP Finished version

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdint.h>

#include <stdbool.h>

#include <stdio.h>

#include "axi\_lw.h"

#include "uart\_function.h"

#include "avalon\_function.h"

int \_\_auto\_semihosting;

void set\_A9\_IRQ\_stack(void);

void config\_GIC(void);

void enable\_A9\_interrupts(void);

int main(void){

printf("Laboratoire: Mesure du temps de reaction \n");

volatile uint32\_t \*constante = (volatile uint32\_t \*)H2F\_BASE\_ADD;

printf("Constante à l'adresse 0xFF200000 de l'AXI: 0x%X\n", \*constante);

constante = (volatile uint32\_t \*)INTERFACE\_BASE\_ADD;

printf("Constante à l'adresse 0xFF210000 de notre interface: 0x%X\n", \*constante);

// Check if the Max10 is connected

if(!Max10\_check\_status()){

printf("Erreur Max10: Configuration non valide.\n");

return -1;

}

// program initialisation

Max10\_init();

disable\_counter();

Leds\_write(LED\_OFF);

Segs7\_init();

uart\_config();

send\_to\_uart("Reaction Time Measurement Application:\r\n - Press KEY1 to start a reaction time measurement.\r\n - Follow the instructions displayed on the Max10\_leds.\r\n - Press KEY0 to stop the measurement.\r\n");

// IRQ initialisation

clear\_irq();

set\_A9\_IRQ\_stack();

config\_GIC();

enable\_A9\_interrupts();

uint32\_t rand\_value;

while(true){

// always display based of the switch value

Seg7\_display(Switchs\_read());

if(Key\_read\_edge(1)){

no\_error();

send\_to\_uart("Reaction time measurement started.\r\nWait for the start symbol on Max10\_leds and press KEY0 as fast as possible.\r\n");

// Waiting symbol

Max10\_write\_square(WAIT\_DISPLAY);

new\_attemps();

// generate a new waiting time between 1s and 4s

rand\_value = generate\_random();

// restart the counter

reset\_counter();

enable\_counter();

// Wait till our counter reach the random value or there is no pression on the key 0 before

while((counter\_current\_value() \* CLOCK\_IN\_SECOND <= rand\_value) && !new\_error());

// if it there wasnt a pression of key0 before

if(!new\_error()){

// so start the measurement

disable\_counter();

start\_game();

// Beginning symbol

Max10\_write\_square(BEGIN\_DISPLAY);

// restart the counter for the measurement

reset\_counter();

enable\_counter();

}

}

}

}

**int\_defines.h :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : int\_defines.h

\* Author : Anthony Convers

\* Date : 27.10.2022

\*

\* Context : ARE lab

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: Some definitions for the interrupt

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Engineer Comments

\* 0.0 27.10.2022 ACS Initial version.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CPU0 0x01 // bit-mask; bit 0 represents cpu0

#define ENABLE 0x1

#define USER\_MODE 0b10000

#define FIQ\_MODE 0b10001

#define IRQ\_MODE 0b10010

#define SVC\_MODE 0b10011

#define ABORT\_MODE 0b10111

#define UNDEF\_MODE 0b11011

#define SYS\_MODE 0b11111

#define INT\_ENABLE 0b01000000

#define INT\_DISABLE 0b11000000

**uart\_function.c :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : uart\_function.c

\* Author : Maillard Patrick

\* Date : 26.01.25

\*

\* Context : Function for UART

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief:

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Student Comments

\* 1.0 26.01.25 MaillardP Finished version

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdint.h>

#include <stdbool.h>

#include <stdio.h>

#include "uart\_function.h"

void uart\_config(){

uint32\_t \*uart\_lcr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + LCR));

uint32\_t \*uart\_fcr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + FCR));

uint32\_t \*uart\_dll = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + RBR\_THR\_DLL));

uint32\_t \*uart\_dlh = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + IER\_DLH));

uint32\_t \*uart\_ier = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + IER\_DLH));

//configure le DLAB de lcr

\*uart\_lcr |= 0x80;

// divisor = l4\_sp\_clk/ (16\*baudrate) =~ 651.04

// Configurer le baud rate (651 = 0x28B)

\*uart\_dll = 0x8B; // LSB

\*uart\_dlh = 0x02; // MSB

// Configurer LCR : 8 bits de données, parité désactivée, 1 bit de stop

\*uart\_lcr = 0x03;

// Activer les FIFO en réception et transmission

\*uart\_fcr = 0x01;

// clear DLAB of lcr

\*uart\_ier = 0x00;

}

void uart\_send\_char(char c) {

uint32\_t \*uart\_thr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + RBR\_THR\_DLL));

uint32\_t \*uart\_lsr = ((volatile unsigned int \*)(UART0\_BASE\_ADDRESS + LSR));

// If the fifo is empty we wait

while (!(\*uart\_lsr & LSR\_THRE\_MASK));

//send our char to the uart

\*uart\_thr = c;

}

void send\_to\_uart(const char\* mystring){

while (\*mystring) {

uart\_send\_char(\*mystring);

mystring++;

}

}

**uart\_function.h :**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* HEIG-VD

\* Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

\* School of Business and Engineering in Canton de Vaud

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* REDS Institute

\* Reconfigurable Embedded Digital Systems

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File : uart\_function.h

\* Author : Maillard Patrick

\* Date :

\*

\* Context : Function for UART

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief:

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Modifications :

\* Ver Date Student Comments

\* 1.0 26.01.25 MaillardP Finished version

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Base address

#define UART0\_BASE\_ADDRESS 0xFFC02000

// UART0 register offset (not all are used)

#define RBR\_THR\_DLL 0x0

#define IER\_DLH 0x4

#define IIR 0x8

#define FCR 0x8

#define LCR 0xC

#define MCR 0x10

#define LSR 0x14

#define MSR 0x18

#define SCR 0x1C

#define SRBR 0x30

#define STHR 0x34

#define FAR 0x70

#define TFR 0x74

#define RFW 0x78

#define USR 0x7C

#define TFL 0x80

#define RFL 0x84

#define SRR 0x88

#define SRTS 0x8C

#define SBCR 0x90

#define SDMAM 0x94

#define SFE 0x98

#define SRT 0x9C

#define STET 0xA0

#define HTX 0xA4

#define DMASA 0xA8

#define CPR 0xF4

#define UCV 0xF8

#define CTR 0xFC

// ACCESS MACROS

#define UART0\_REG(\_x\_) (volatile uint32\_t \*)(AXI\_LW\_HPS\_FPGA\_BASE\_ADD + \_x\_)

// MASK

#define LSR\_THRE\_MASK 0x20

// uart\_config function : Initialise the UART module so it can send our data correctly

// Parameter : None

// Return : None

void uart\_config();

// uart\_send\_char : Send a character to the UART

// Parameter : "c" = the character to be sent

// Return : None

void uart\_send\_char(char c);

// send\_to\_uart : Send a string to the UART

// Parameter : "c" = the string to be sent

// Return : None

void send\_to\_uart(const char\* mystring);

**avl\_user\_interface.vhdl (v2.1):**

-----------------------------------------------------------------------------------

-- HEIG-VD ////////////////////////////////////////////////////////////////////////

-- Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud

-- School of Business and Engineering in Canton de Vaud

-----------------------------------------------------------------------------------

-- REDS Institute /////////////////////////////////////////////////////////////////

-- Reconfigurable Embedded Digital Systems

-----------------------------------------------------------------------------------

--

-- File : avl\_user\_interface.vhd

-- Author : Guillaume Gonin

-- Date : 20.12.2024

--

-- Context : Avalon user interface

--

-----------------------------------------------------------------------------------

-- Description :

--

-----------------------------------------------------------------------------------

-- Dependencies :

--

-----------------------------------------------------------------------------------

-- Modifications :

-- Ver Date Engineer Comments

-- 0.0 18.12.2024 UB/GoninG Initial version

-- 1.0 07.01.2025 GoninG Adapted to my needs

-- 1.1 07.01.2025 GoninG Added counter

-- 1.2 10.01.2025 GoninG Added irq

-- 1.3 10.01.2025 GoninG No compile errors

-- 1.4 10.01.2025 GoninG Serial transmitter done

-- 1.5 10.01.2025 GoninG Renamed process and some error resolved

-- 2.0 17.01.2025 GoninG Serial transmitter and simulated (simulate didn't found out other errors)

-- 2.1 24.01.2025 MaillardP button edge detection inversed

-----------------------------------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity avl\_user\_interface is

port(

-- Avalon bus

avl\_clk\_i : in std\_logic;

avl\_reset\_i : in std\_logic;

avl\_address\_i : in std\_logic\_vector(13 downto 0);

avl\_byteenable\_i : in std\_logic\_vector(3 downto 0);

avl\_write\_i : in std\_logic;

avl\_writedata\_i : in std\_logic\_vector(31 downto 0);

avl\_read\_i : in std\_logic;

avl\_readdatavalid\_o : out std\_logic;

avl\_readdata\_o : out std\_logic\_vector(31 downto 0);

avl\_waitrequest\_o : out std\_logic;

avl\_irq\_o : out std\_logic;

-- User interface

button\_i : in std\_logic\_vector(3 downto 0);

switch\_i : in std\_logic\_vector(9 downto 0);

led\_o : out std\_logic\_vector(9 downto 0);

hex0\_o : out std\_logic\_vector(6 downto 0);

hex1\_o : out std\_logic\_vector(6 downto 0);

hex2\_o : out std\_logic\_vector(6 downto 0);

hex3\_o : out std\_logic\_vector(6 downto 0);

-- Con 80p interface

serial\_data\_o : out std\_logic;

con\_80p\_status\_i : in std\_logic\_vector(1 downto 0)

);

end avl\_user\_interface;

architecture rtl of avl\_user\_interface is

--| Components declaration |-----------------------------------------------------

--| Constants declarations |-----------------------------------------------------

CONSTANT INTERFACE\_ID\_C : STD\_LOGIC\_VECTOR(31 DOWNTO 0) := x"12345678";

CONSTANT RESERVED\_VAL\_C : STD\_LOGIC\_VECTOR(31 DOWNTO 0) := x"DEADBEEF";

CONSTANT AVL\_CLOCK\_RATE\_C : UNSIGNED(27 DOWNTO 0) := x"2FAF080"; -- 50MHz = 0x2FAF080

CONSTANT SERIAL\_TRANSMITTER\_BAUDRATE\_C : UNSIGNED(15 DOWNTO 0) := x"2580"; -- 9600 = 0x2580

--| Signals declarations |-----------------------------------------------------

-- I/O

SIGNAL led\_reg\_s : STD\_LOGIC\_VECTOR(9 DOWNTO 0);

SIGNAL hex0\_reg\_s : STD\_LOGIC\_VECTOR(6 DOWNTO 0);

SIGNAL hex1\_reg\_s : STD\_LOGIC\_VECTOR(6 DOWNTO 0);

SIGNAL hex2\_reg\_s : STD\_LOGIC\_VECTOR(6 DOWNTO 0);

SIGNAL hex3\_reg\_s : STD\_LOGIC\_VECTOR(6 DOWNTO 0);

SIGNAL switches\_s : STD\_LOGIC\_VECTOR(9 DOWNTO 0);

SIGNAL button\_s : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

-- Con 80p interface

SIGNAL serial\_data\_s : STD\_LOGIC;

SIGNAL f\_ready\_s : STD\_LOGIC;

SIGNAL con\_80p\_status\_s : STD\_LOGIC\_VECTOR(1 DOWNTO 0);

-- Avalon Interface

SIGNAL readdatavalid\_next\_s : STD\_LOGIC;

SIGNAL readdatavalid\_reg\_s : STD\_LOGIC;

SIGNAL readdata\_user\_s : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL readdata\_reg\_s : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL irq\_s : STD\_LOGIC;

-- Counter

SIGNAL counter\_value\_next\_s : UNSIGNED(31 DOWNTO 0);

SIGNAL counter\_value\_reg\_s : UNSIGNED(31 DOWNTO 0);

SIGNAL enable\_counter\_s : STD\_LOGIC;

SIGNAL reset\_counter\_s : STD\_LOGIC;

-- Internal uses

SIGNAL last\_button0\_state\_s : STD\_LOGIC;

SIGNAL button0\_s : STD\_LOGIC;

SIGNAL button0\_irq\_s : STD\_LOGIC;

SIGNAL clear\_irq\_s : STD\_LOGIC;

SIGNAL mask\_irq\_s : STD\_LOGIC;

SIGNAL clock\_count\_s : UNSIGNED(12 DOWNTO 0); -- 5208 needs 13 bits

SIGNAL bits\_count\_s : UNSIGNED(4 DOWNTO 0); -- 22 bits at max

SIGNAL pulse\_serial\_s : STD\_LOGIC;

SIGNAL data\_reg\_s : STD\_LOGIC\_VECTOR(19 DOWNTO 0);

SIGNAL data\_user\_s : STD\_LOGIC\_VECTOR(19 DOWNTO 0);

begin

-- Input signals

con\_80p\_status\_s <= con\_80p\_status\_i;

button\_s <= button\_i;

switches\_s <= switch\_i;

-- Output signals

avl\_readdatavalid\_o <= readdatavalid\_reg\_s;

avl\_readdata\_o <= readdata\_reg\_s;

avl\_irq\_o <= irq\_s;

led\_o <= led\_reg\_s;

hex0\_o <= hex0\_reg\_s;

hex1\_o <= hex1\_reg\_s;

hex2\_o <= hex2\_reg\_s;

hex3\_o <= hex3\_reg\_s;

serial\_data\_o <= serial\_data\_s;

-- Read access part

read\_decoder\_p : PROCESS (ALL)

BEGIN

readdatavalid\_next\_s <= '0'; --valeur par defaut

readdata\_user\_s <= (OTHERS => '0'); --valeur par defaut

IF avl\_read\_i = '1' THEN

readdatavalid\_next\_s <= '1';

CASE (to\_integer(unsigned(avl\_address\_i))) IS

WHEN 0 =>

readdata\_user\_s <= INTERFACE\_ID\_C;

WHEN 1 =>

readdata\_user\_s(3 DOWNTO 0) <= button\_s;

WHEN 2 =>

readdata\_user\_s(9 DOWNTO 0) <= switches\_s;

WHEN 3 =>

readdata\_user\_s(9 DOWNTO 0) <= led\_reg\_s;

WHEN 4 =>

readdata\_user\_s(6 DOWNTO 0) <= hex0\_reg\_s;

readdata\_user\_s(13 DOWNTO 7) <= hex1\_reg\_s;

readdata\_user\_s(20 DOWNTO 14) <= hex2\_reg\_s;

readdata\_user\_s(27 DOWNTO 21) <= hex3\_reg\_s;

WHEN 5 =>

readdata\_user\_s(1 DOWNTO 0) <= con\_80p\_status\_s;

readdata\_user\_s(2) <= f\_ready\_s;

WHEN 6 =>

readdata\_user\_s(31 DOWNTO 0) <= std\_logic\_vector(counter\_value\_reg\_s);

WHEN 7 =>

readdata\_user\_s(0) <= irq\_s;

WHEN OTHERS =>

readdata\_user\_s <= RESERVED\_VAL\_C;

END CASE;

END IF;

END PROCESS;

-- Read register process

read\_register\_p : PROCESS (avl\_reset\_i, avl\_clk\_i)

BEGIN

IF avl\_reset\_i = '1' THEN

readdatavalid\_reg\_s <= '0';

readdata\_reg\_s <= (OTHERS => '0');

ELSIF rising\_edge(avl\_clk\_i) THEN

readdatavalid\_reg\_s <= readdatavalid\_next\_s;

readdata\_reg\_s <= readdata\_user\_s;

END IF;

END PROCESS;

-- Write access part

write\_register\_p : PROCESS (

avl\_reset\_i,

avl\_clk\_i

)

BEGIN

IF avl\_reset\_i = '1' THEN

led\_reg\_s <= (OTHERS => '0');

hex0\_reg\_s <= (OTHERS => '0');

hex1\_reg\_s <= (OTHERS => '0');

hex2\_reg\_s <= (OTHERS => '0');

hex3\_reg\_s <= (OTHERS => '0');

reset\_counter\_s <= '0';

enable\_counter\_s <= '0';

clear\_irq\_s <= '0';

mask\_irq\_s <= '0';

pulse\_serial\_s <= '0';

ELSIF rising\_edge(avl\_clk\_i) THEN

reset\_counter\_s <= '0'; --valeur par defaut

clear\_irq\_s <= '0'; --valeur par defaut

pulse\_serial\_s <= '0'; --valeur par defaut

IF avl\_write\_i = '1' THEN

CASE (to\_integer(unsigned(avl\_address\_i))) IS

WHEN 3 =>

led\_reg\_s <= avl\_writedata\_i(9 DOWNTO 0);

WHEN 4 =>

hex0\_reg\_s <= avl\_writedata\_i(6 DOWNTO 0);

hex1\_reg\_s <= avl\_writedata\_i(13 DOWNTO 7);

hex2\_reg\_s <= avl\_writedata\_i(20 DOWNTO 14);

hex3\_reg\_s <= avl\_writedata\_i(27 DOWNTO 21);

WHEN 5 =>

IF(f\_ready\_s = '1') THEN

data\_user\_s <= avl\_writedata\_i(19 DOWNTO 0);

pulse\_serial\_s <= '1';

END IF;

WHEN 6 =>

reset\_counter\_s <= avl\_writedata\_i(0);

enable\_counter\_s <= avl\_writedata\_i(1);

WHEN 7 =>

clear\_irq\_s <= avl\_writedata\_i(0);

mask\_irq\_s <= avl\_writedata\_i(1);

WHEN OTHERS =>

NULL;

END CASE;

END IF;

END IF;

END PROCESS;

-- Counter process

counter\_p : PROCESS (avl\_reset\_i, avl\_clk\_i)

BEGIN

IF avl\_reset\_i = '1' THEN

counter\_value\_reg\_s <= (OTHERS => '0');

counter\_value\_next\_s <= (OTHERS => '0');

ELSIF rising\_edge(avl\_clk\_i) THEN

counter\_value\_reg\_s <= counter\_value\_next\_s;

IF reset\_counter\_s = '1' THEN

counter\_value\_next\_s <= (OTHERS => '0');

ELSIF enable\_counter\_s = '0' THEN

counter\_value\_next\_s <= counter\_value\_next\_s;

ELSIF counter\_value\_next\_s = INTEGER'high THEN -- "FF..FF"

counter\_value\_next\_s <= counter\_value\_next\_s; -- counter\_value\_next\_s + 1 would equals 0

ELSE

counter\_value\_next\_s <= counter\_value\_next\_s + 1;

END IF;

END IF;

END PROCESS;

-- Button 0 reg

button0\_register\_p : PROCESS (avl\_reset\_i, avl\_clk\_i)

BEGIN

IF avl\_reset\_i = '1' THEN

last\_button0\_state\_s <= '0';

button0\_s <= '0';

ELSIF rising\_edge(avl\_clk\_i) THEN

button0\_s <= button\_s(0);

last\_button0\_state\_s <= button0\_s;

END IF;

END PROCESS;

-- Irq process

irq\_p : PROCESS (avl\_reset\_i, avl\_clk\_i)

BEGIN

IF avl\_reset\_i = '1' THEN

irq\_s <= '0';

button0\_irq\_s <= '0';

ELSIF rising\_edge(avl\_clk\_i) THEN

IF clear\_irq\_s = '1' THEN

irq\_s <= '0';

button0\_irq\_s <= '0';

ELSIF mask\_irq\_s = '0' AND button0\_s = '0' AND last\_button0\_state\_s = '1' THEN

irq\_s <= '1';

button0\_irq\_s <= '1';

END IF;

END IF;

END PROCESS;

-- Serial transmitter process

serial\_transmitter\_p : PROCESS (avl\_reset\_i, avl\_clk\_i)

BEGIN

IF avl\_reset\_i = '1' THEN

clock\_count\_s <= (OTHERS => '0');

bits\_count\_s <= (OTHERS => '0');

f\_ready\_s <= '1';

data\_reg\_s <= (OTHERS => '1');

serial\_data\_s <= '1';

ELSIF rising\_edge(avl\_clk\_i) THEN

IF bits\_count\_s = 22 AND clock\_count\_s = (AVL\_CLOCK\_RATE\_C/SERIAL\_TRANSMITTER\_BAUDRATE\_C)-1 THEN -- all bits written and last bit holded all the time needed

f\_ready\_s <= '1';

clock\_count\_s <= (OTHERS => '0');

bits\_count\_s <= (OTHERS => '0');

ELSIF pulse\_serial\_s = '1' THEN

bits\_count\_s <= "00001"; -- start bit now written

serial\_data\_s <= '0';

data\_reg\_s <= data\_user\_s;

f\_ready\_s <= '0';

ELSIF f\_ready\_s = '1' THEN

clock\_count\_s <= (OTHERS => '0');

ELSIF clock\_count\_s = (AVL\_CLOCK\_RATE\_C/SERIAL\_TRANSMITTER\_BAUDRATE\_C)-1 THEN -- when clock\_count is avl\_clk/baudrate times we write new val

clock\_count\_s <= (OTHERS => '0');

bits\_count\_s <= bits\_count\_s + 1;

serial\_data\_s <= data\_reg\_s(19); -- MSB

data\_reg\_s(19 DOWNTO 1) <= data\_reg\_s(18 DOWNTO 0);

data\_reg\_s(0) <= '1'; -- data\_ << 1 and append 1 in LSB

ELSE

clock\_count\_s <= clock\_count\_s + 1;

END IF;

END IF;

END PROCESS;

end rtl;