ARE - Labo 5

Pedro Alves da Silva Urs Behrmann

Contents

Introduction

Dans ce laboratoire, nous allons concevoir une interface permettant d'effectuer la lecture de différentes chaînes de caractères générées par la board DE1-SoC. La génération de ces dernières peuvent être effectués à différentes fréquences (1Hz, 1KHz, 100KHz, 1MHz), et de façon manuelle ou automatique.

Lors de la première partie, les lectures seront effectuées de façon non-fiable, ce qui permettra l'accès aux différents blocs de la chaîne alors que l'interface est en train de remplacer les valeurs des blocks, et de remarquer que des erreurs de lecture risque d'arriver. La deuxième partie nous permettra d'implémenter un méchanisme permettant d'éviter ces lectures éronnées.

Afin de vérifier si une lecture correcte a eu lieu, un byte de checksum est utilisé. La lecture est considérée comme correcte lorsque la somme des valeurs ASCII de chacun des caractères lus plus le byte de checksum, le tout modulo 256, vaut 0.

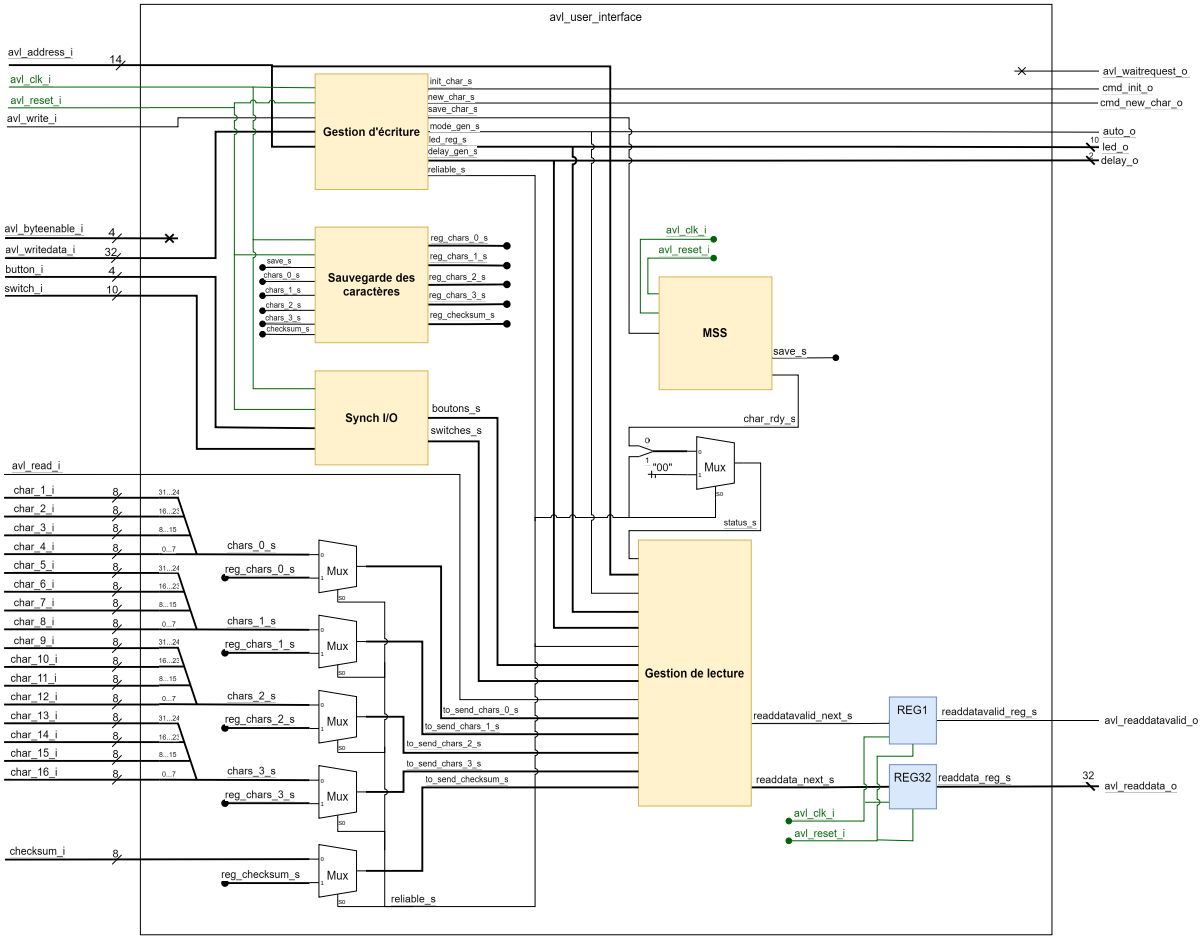
Si une lecture n'a pas été éffectuée correctement, un message à l'écran s'affiche, et le nombre de fois qu'une lecture erronée a été faite depuis le lancement du programme est également affiché.

Analyse

Plan d'adressage

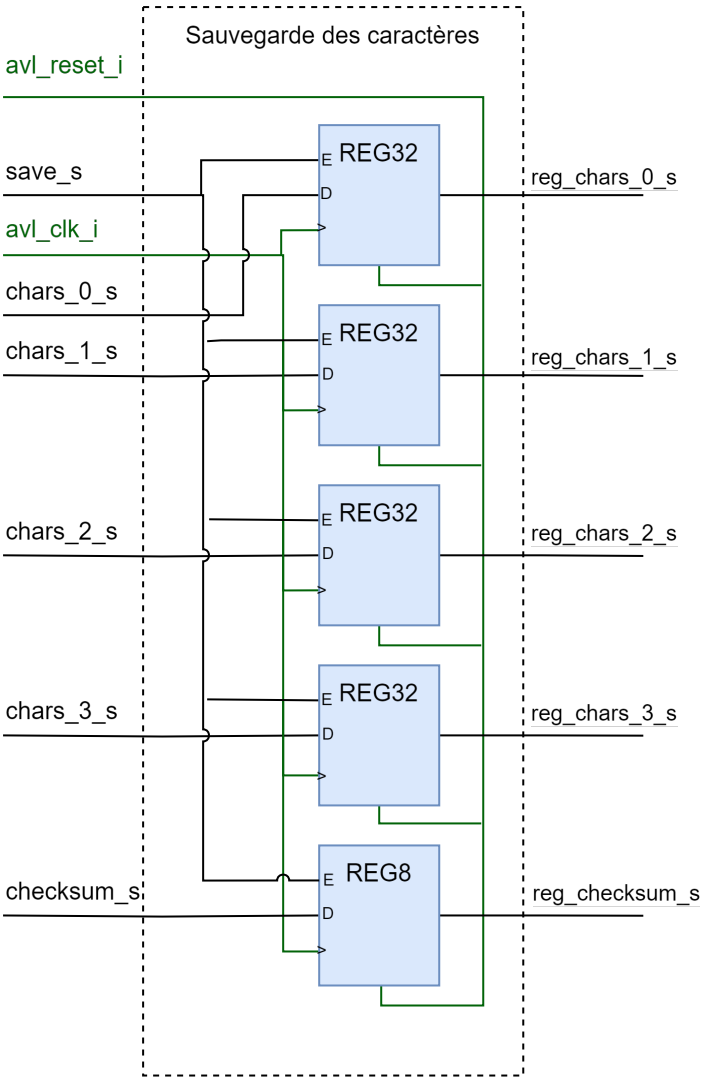
Address (offset)	Read	Write
0x00	[31..0] Interface user ID	reserved
0x04	[31..4] "0..0"; [3..0] buttons	reserved
0x08	[31..10] "0..0"; [9..0] switches	reserved
0x0C	[31..10] "0..0"; [9..0] leds	[31..10] reserved; [9..0] leds
0x10	[31..2] "0..0"; [1..0] status	[31..5] reserved; [4] new_char [3..1] reserved; [0] init_char
0x14	[31..5] "0..0"; [4] mode_gen; [3..2] "0..0"; [1..0] delay_gen	[31..5] reserved; [4] mode_gen [3..2] reserved; [1..0] delay_gen
0x18	[31..0] "0..0"	[31..1] reserved; [0] save_char
0x1C	[31..1] "0..0"; [0] reliable	[31..1] reserved; [0] reliable
0x20	[31..24] char_2; [23..16] char_3; [15..8] char_4; [7..0] char_4	reserved
0x24	[31..24] char_5; [23..16] char_6; [15..8] char_7; [7..0] char_8	reserved
0x28	[31..24] char_9; [23..16] char_10; [15..8] char_11; [7..0] char_12	reserved
0x2C	[31..24] char_13; [23..16] char_14; [15..8] char_15; [7..0] char_16	reserved
0x30	[31..8] "0..0"; [7..0] checksum	reserved
0x34 … 0xFFFC	[31..0] "0..0"	reserved

Schéma bloc de l'interface



Sauvegarde des caractères

Ce bloc sert simplement à sauvegarder les caractères automatiquement générés reçus en entrée. La seule entrée synchrone utilisée dans ce bloc est `save_s`, sortant de la MSS.



Gestion d'écriture

Ce bloc sert à sauvegarder les données à écrire dans des différents registres. L'idée c'est qu'il s'en charge et du "décodage d'adresses", et de l'extraction des données venant du bus Avalon. De ce fait, au lieu de répéter le même schéma que pour le bloc "Sauvegarde des caractères", car ce bloc est également compris de registres, on donne ici les équations pour l'enable de chacun des registres.

Pour rappel, les entrées synchrones dans ce bloc sont: `avl_writedata_i`, `avl_address_i`, et `avl_write_i`.

- `init_char_s`: `avl_write_i AND avl_writedata_i(0) AND (avl_address_i = 4)`
- `new_char_s`: `avl_write_i AND avl_writedata_i(4) AND (avl_address_i = 4)`
- `save_char_s`: `avl_write_i AND avl_writedata_i(0) AND (avl_address_i = 6)`
- `mode_gen_s`: `avl_write_i AND avl_writedata_i(4) AND (avl_address_i = 5)`
- `led_reg_s`: `avl_writedata_i(9 downto 0)` si `avl_write_i AND (avl_address_i = 3)`
- `delay_gen_s`: `avl_writedata_i(1 downto 0)` si `avl_write_i AND (avl_address_i = 5)`
- `reliable_s`: `avl_write_i AND avl_writedata_i(0) AND (avl_address_i = 7)`

Synch I/O

Tout comme les 2 précédents blocs, ce bloc permet également simplement de sauvegarder des données dans des registres. Toutefois les registres dans ce bloc n'ont pas d'enable, et se content de synchroniser

les 2 entrées à chaque coup de clock.

Gestion de lecture

Ce bloc agit comme un multiplexeur pour savoir quoi enregistrer dans le signal utilisé pour sortie les données lues par le CPU. On utilise l'adresse reçue comme bits de sélection; 14 au total. Du coup, dans cette section, nous décrivons ce qu'on écrit dans le signal dans chacun des cas.

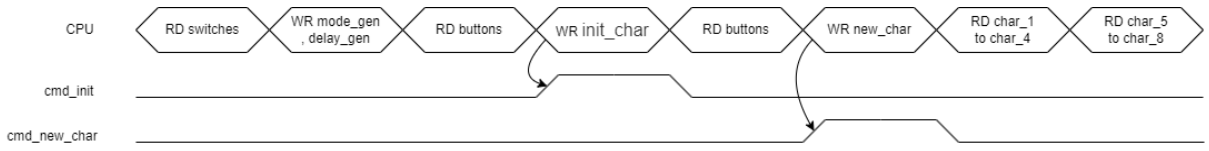
Ce bloc aurait pu être amélioré, en utilisant simplement les 4 least-significant bits de l'adresse, car nous n'avons que 13 adresses différentes à affecter, et on pourrait vérifier qu'on se trouve bien au tout début de l'espace mémoire de l'interface (`avl_address_i(13 downto 4) = "00...0"`) avant d'affecter le signal utilisé en sortie.

- `avl_address_i = 0`: ID d'interface (0x12345678)
- `avl_address_i = 1`: état actuel des boutons
- `avl_address_i = 2`: état actuel des switches
- `avl_address_i = 3`: état actuel des LEDs
- `avl_address_i = 5`: status de l'interface
- `avl_address_i = 6`: concaténation du mode de génération (bit 4) avec le délai de génération (bits 1 à 0)
- `avl_address_i = 7`: informe si la lecture est fiable
- `avl_address_i = 8`: premier buffer de 4 caractères
- `avl_address_i = 9`: deuxième buffer de 4 caractères
- `avl_address_i = 10`: troisième buffer de 4 caractères
- `avl_address_i = 11`: quatrième buffer de 4 caractères
- `avl_address_i = 12`: checksum (bits 7 à 0)
- Autre valeur: `"00...0"`

Chronogramme de fonctionnement

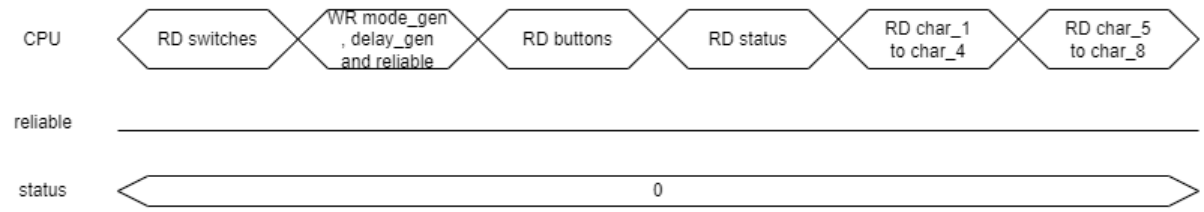
Fonctionnement manuel

Pour le fonctionnement manuel, le CPU envoie des commandes à l'interface pour reseter l'interface, générer les prochains caractères ou lire les caractères.

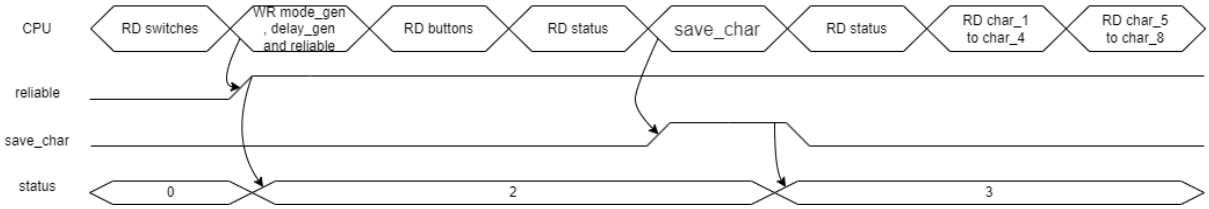


Fonctionnement automatique non-fiable

Dans le fonctionnement automatique non-fiable, l'interface génère les caractères automatiquement selon la fréquence définie ('delay_gen'). Le CPU peut lire les caractères à tout moment, mais n'a aucune garantie que les prochaines charactères sont de la même séquence que les précédents. Seulement avec vérification du checksum, le CPU peut savoir si les caractères lus sont corrects.

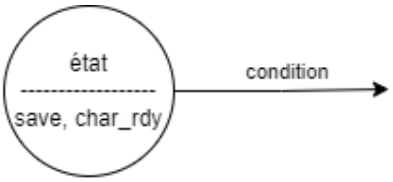


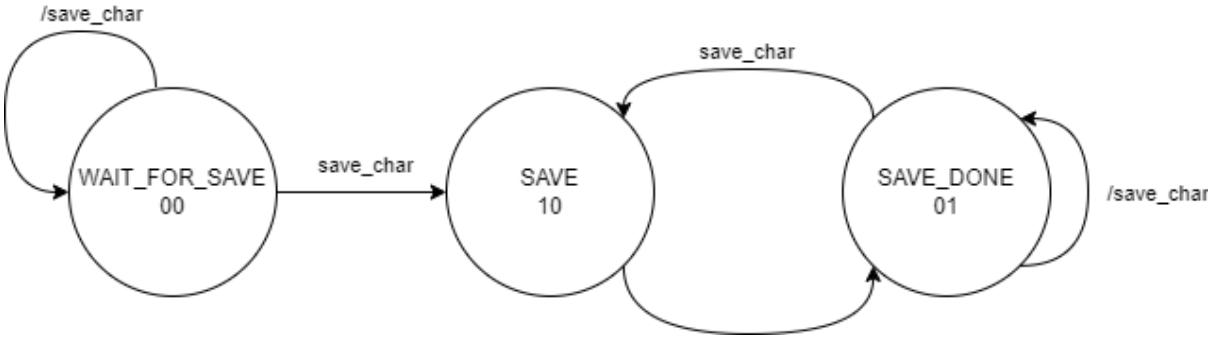
Fonctionnement automatique fiable



MSS

Pour la partie 2, nous avons ajouté une MSS qui permet de cadencer la sauvegarde et libération de la lecture des caractères. Cette MSS est composée de 3 états: `WAIT_FOR_SAVE`, `SAVE` et `SAVE_DONE`. Lorsque l'interface est en mode automatique, la MSS attend qu'une sauvegarde soit demandée. Lorsque la sauvegarde est demandée, l'interface sauvegarde les caractères dans un registre. Au prochain cycle d'horloge, l'interface informe le CPU via le registre `status(0)` que la sauvegarde est terminée.





Code

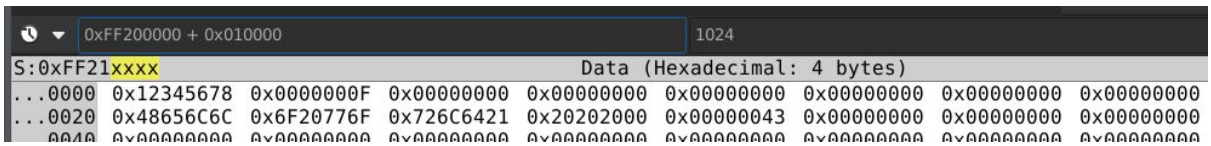
La base de notre code est très semblable au code des laboratoires précédents. Toutefois, elle a été adapté avec les nouvelles adresses et masques à utiliser.

L'interface utilisant le bus Avalon, les adresses qui nous intéressent commencent à `0xFF20 0000`, soit l'adresse de base pour l'AXI lightweight HPS-to-FPGA. La zone mémoire pour notre interface se trouve à l'offset `0x0001 0000`, et nous disposons de 64KiB de mémoire. De ce fait, notre interface interagit avec l'espace mémoire, offset, `0x0001 0000` à `0x0001 FFFF`.

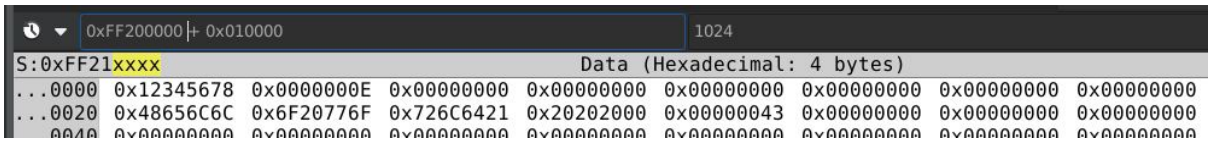
Le code va lire l'ID du design standard à la première adresse de cet espace mémoire (offset `0x0`).

Accès mémoire

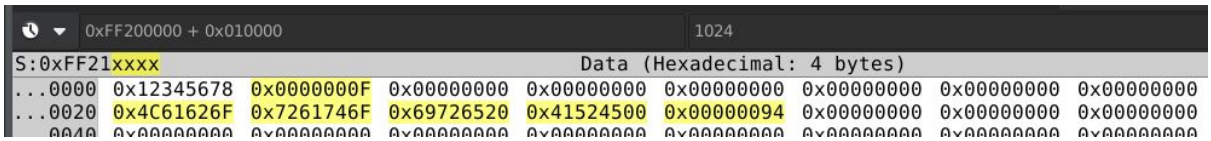
Lorsqu'on lance l'application, on retrouve la mémoire à l'état ci-dessous. L'ID de l'interface y est visible, la valeur des keys est lu (en active low), et on y retrouve également la première chaîne de caractères lue. Dans notre cas, "Hello world!".



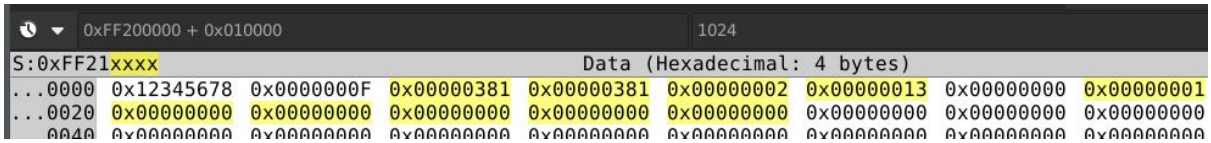
La pression sur le Key0, qui réinitialise le générateur, n'a pas d'impacte sur la mémoire, car la chaîne de caractères qu'on y retrouve est déjà la chaîne de caractères initiale.



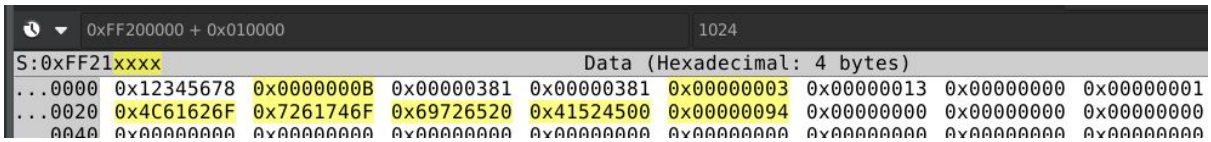
Lorsqu'on appuie sur le Key1, on remarque qu'une nouvelle chaîne de caractères est générée.



On laisse tourner l'application dans le mode automatique, avec la fréquence maximale. Tant qu'on n'appuie pas sur le Key2, utilisé pour effectuer la lecture des chaînes de caractères, le buffer de caractères qu'on retrouve en mémoire est rempli de zéros.



Dès qu'on appuie sur le Key2, la demande de lecture fiable est envoyée à l'interface, et la chaîne peut être retrouvée peu après en mémoire.



Tests

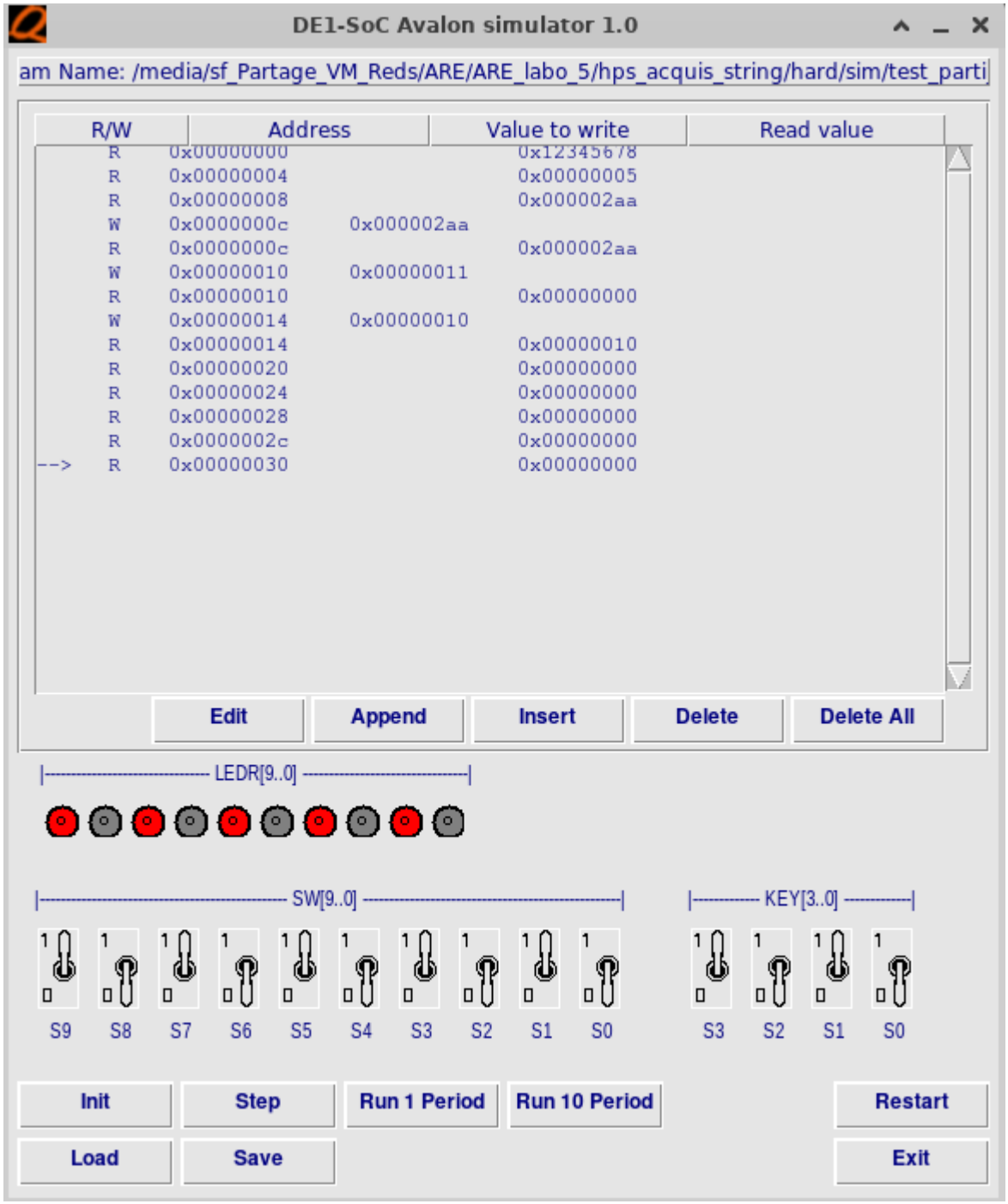
Partie 1

Simulation

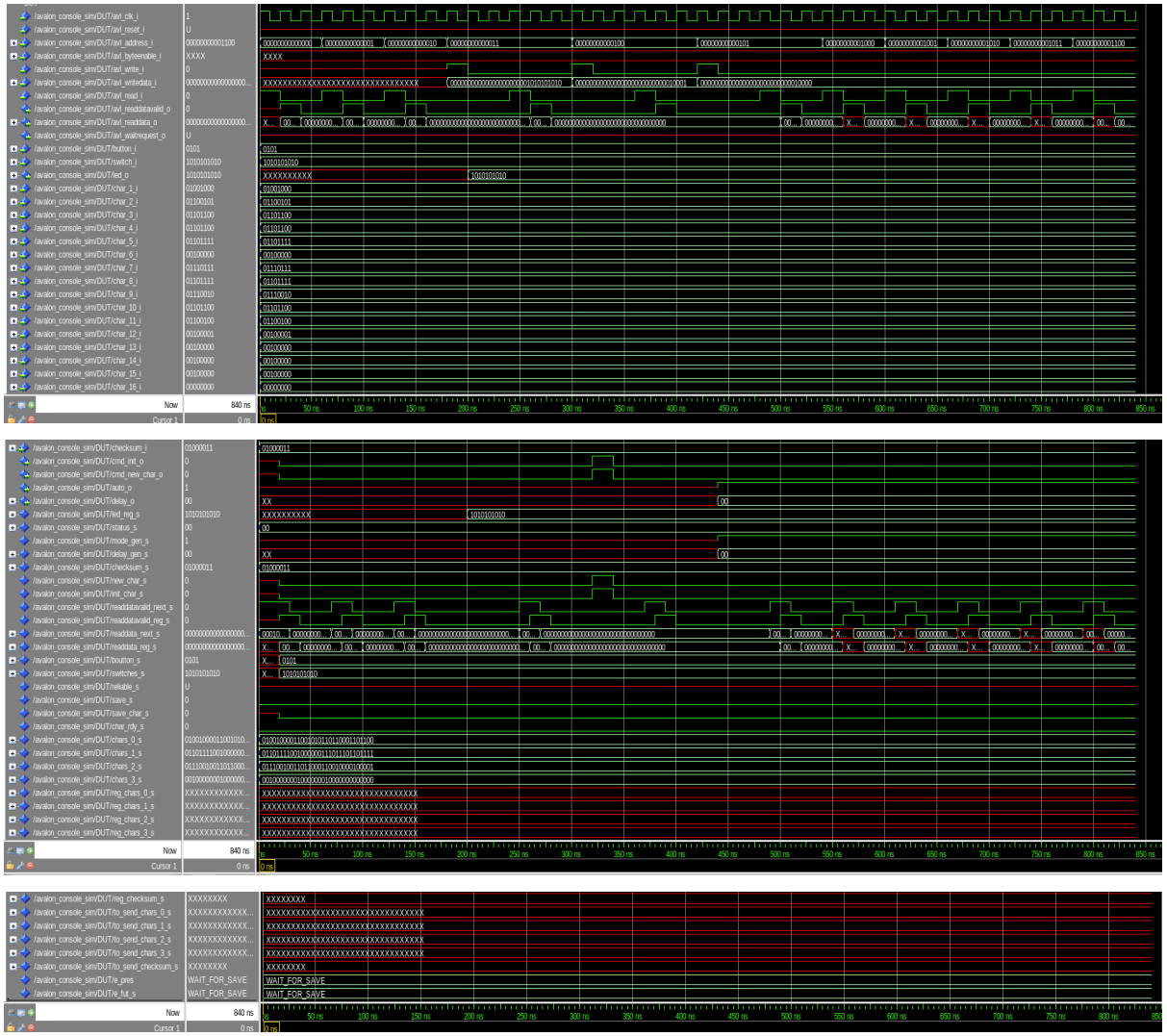
Pour la partie 1, nous avons effectué en premier des tests avec le test bench fourni. Nous avons créé une série de commandes que le CPU peut envoyer à l'interface. Nous avons testé toutes les lectures et écritures décrites dans la description du laboratoire.

Commande	Description
`Read 0`	Lecture de l'interface user ID
`Read 4`	Lecture des 4 buttons
`Read 8`	Lecture des 10 switches
`Write 12 682`	Ecriture de 682 sur les 10 LEDs (0b1010101010)
`Read 12`	Lecture des 10 LEDs
`Write 16 17`	Ecriture dans new_char [4] et init_char [0]
`Read 16`	Lecture du status
`Write 20 16`	Ecriture dans mode_gen [4] et delay_gen [1-0]
`Read 20`	Lecture du mode_gen [4] et de delay_gen [1-0]
`Read 32`	Lecture des char 1 à 4
`Read 36`	Lecture des char 5 à 8
`Read 40`	Lecture des char 9 à 12
`Read 44`	Lecture des char 13 à 16
`Read 48`	Lecture du checksum [7-0] 48

Ici, un screenshot de l'interface de simulation:



Et les chronogrammes:



Cette simulation nous a permis de vérifier que l'interface réagite correctement aux commandes du CPU.

Tests sur la carte

> **NOTE:** Les tests ont été effectués de façon séquentielle. Du coup, si nous effectuons le test "Set du switch 7 à 1 active le mode automatique", alors le mode automatique reste actif jusqu'à-ce qu'on le désactive avec "Set du switch 7 à 0 active le mode automatique".

Pour la partie 1, nous avons effectué les tests ci-dessous sur la carte. Tous les tests mentionnés ont été effectués avec succès.

Test effectué
Le programme affiche les 2 IDs attendus
Pression de la Key0 réinitialise le générateur réinitialise le générateur à "Hello world!"
Pression sur Key2 affiche une lecture
La lecture affiche les caractères dans le bon ordre ("Hello world!" comparé à "lleHow o!dlr")
Calcul d'intégrité est correct (somme des caractères + valeur de checksum)
Maintient de la Key2 affiche la même lecture continuellement
Maintient de la Key2 + pression de la Key0 réinitialise le générateur à "Hello world!"
Maintient de la Key2 + pression de la Key1 affiche une lecture avec une string différente
Maintient de la Key2 + maintient de la Key1 affiche une seule lecture différente
Set du switch 7 à 1 active le mode automatique
Affichage des lectures affiche les bonnes informations en cas de succès
Affichage des lectures affiche les bonnes informations en cas d'erreur

Test effectué
Le compteur des erreurs incrémente à chaque erreur
Incrém. de la fréq. est effectuée, et le nombre de strings différentes par seconde augmente
Changement de fréquence reflet le nombre d'erreur perçues
Décr. de la fréq., à partir de la valeur maximale jusqu'à minimale, est effectuée correctement
Set du switch 7 à 0 désactive le mode automatique
Pression de la Key0 lorsque le mode automatique est actif réinitialise le générateur

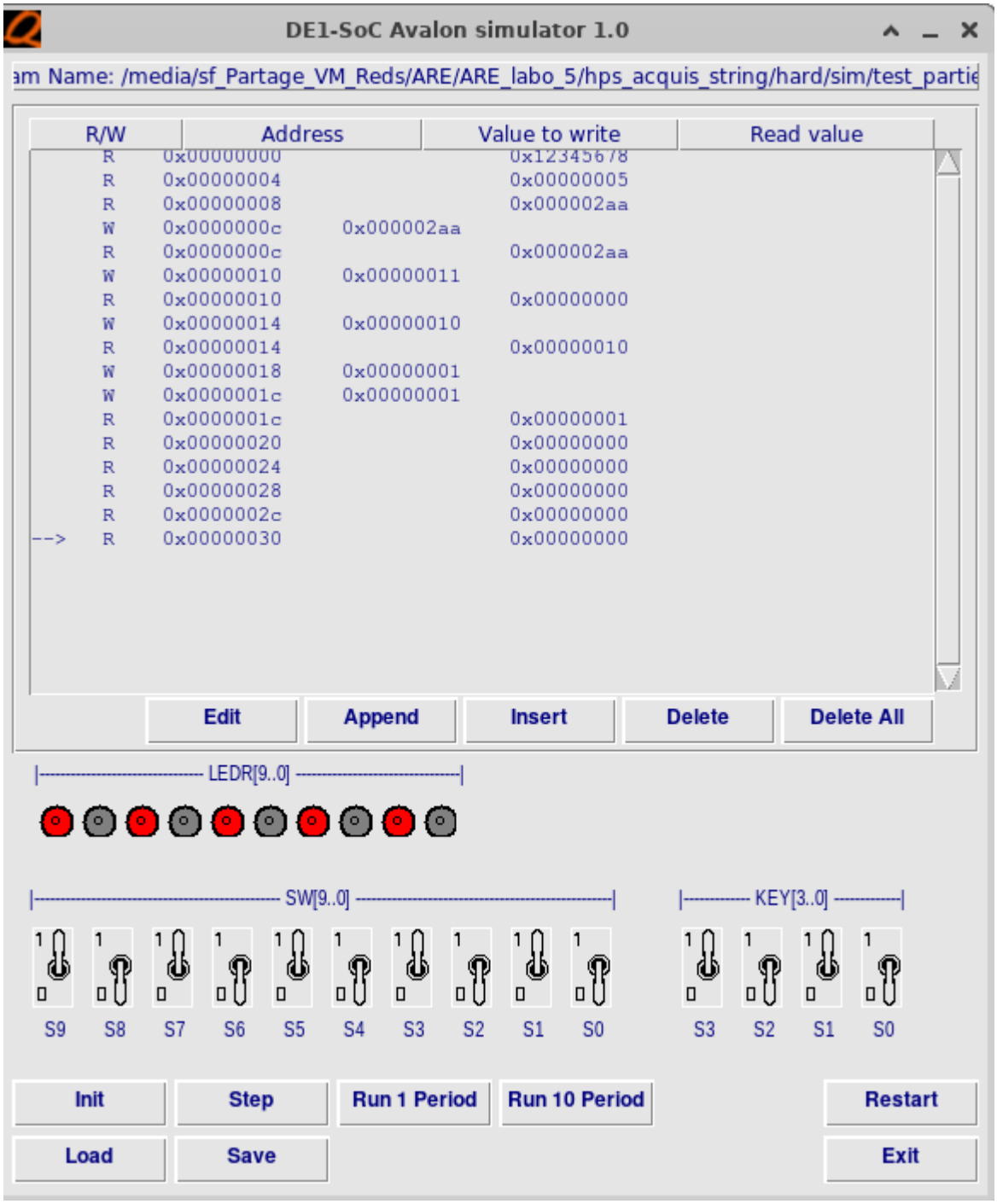
Partie 2

Simulation

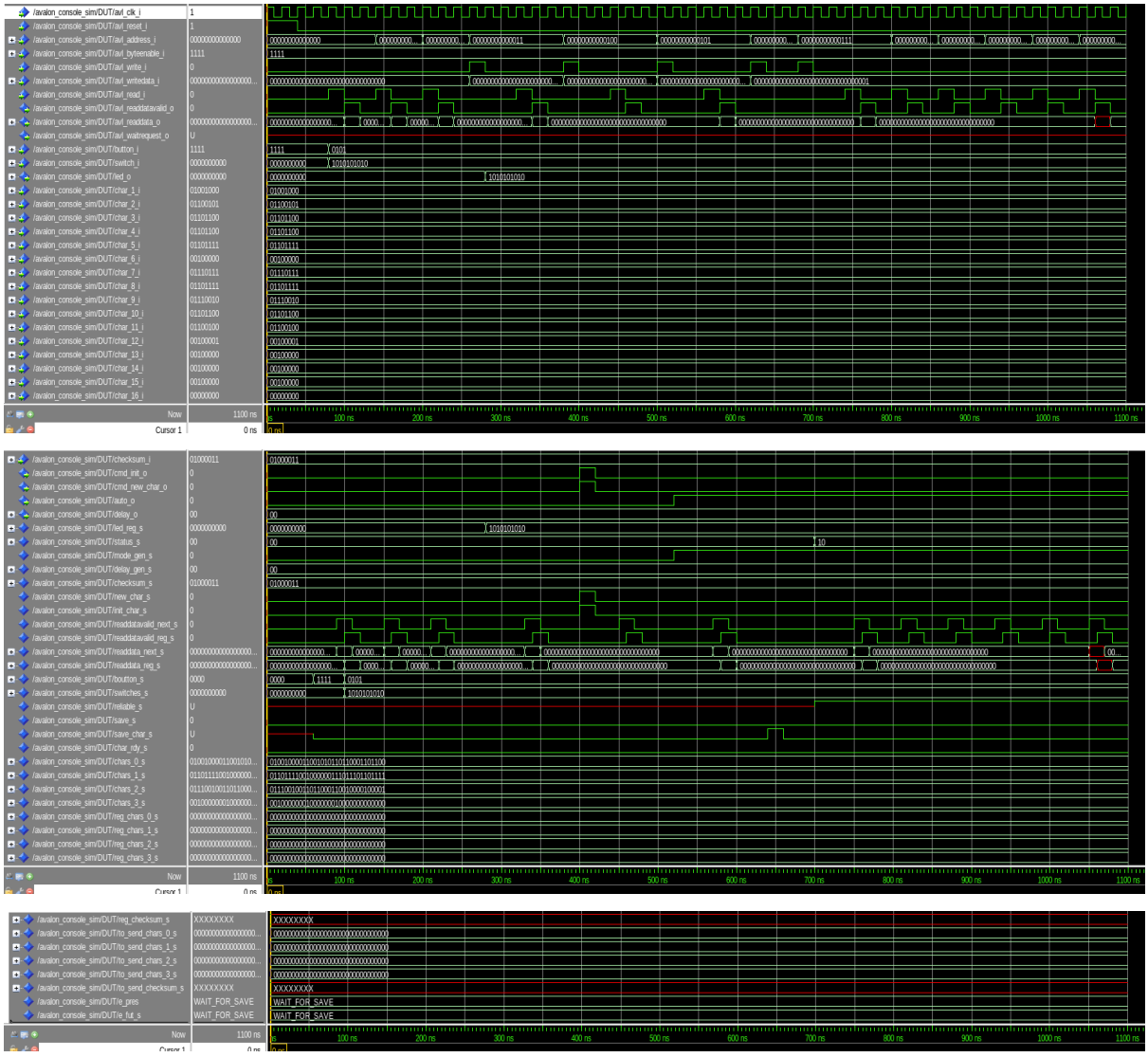
Pour la simulation de la partie 2, nous avons effectué les mêmes tests que pour la partie 1, mais en ajoutant les tests pour les nouvelles fonctionnalités. Voici les commandes utilisées:

Commande	Description
`Read 0`	Lecture de l'interface user ID
`Read 4`	Lecture des 4 buttons
`Read 8`	Lecture des 10 switches
`Write 12 682`	Ecriture de 682 sur les 10 LEDs (0b1010101010)
`Read 12`	Lecture des 10 LEDs
`Write 16 17`	Ecriture dans new_char [4] et init_char [0]
`Read 16`	Lecture du status
`Write 20 16`	Ecriture dans mode_gen [4] et delay_gen [1-0]
`Read 20`	Lecture du mode_gen [4] et de delay_gen [1-0]
`Write 24 1`	Ecriture dans save_char [0]
`Write 28 1`	Ecriture dans reliable [0]
`Read 28`	Lecture de reliable [0]
`Read 32`	Lecture des char 1 à 4
`Read 36`	Lecture des char 5 à 8
`Read 40`	Lecture des char 9 à 12
`Read 44`	Lecture des char 13 à 16
`Read 48`	Lecture du checksum [7-0] 48

Ici, un screenshot de l'interface de simulation:



Et les chronogrammes:



Tests sur la carte

Pour la partie 2, nous avons re-effectué les tests de la partie 1, ainsi que les tests suivants:

Test effectué
Set du switch 7 à 1 active le mode automatique
Set du switch 0 à 1 active le mode fiable
Fréquence plus basse (1Hz) génère des mots dans la bonne fréquence
Pression sur Key2 affiche une lecture sans erreurs
Maintient sur Key2 affiche une lecture sans erreurs
Incrém. de la fréq. est effectuée, et le nombre de lectures incorrectes reste à 0
Set du switch 0 à 0 désactive le mode fiable
Maintient de la Key2 affiche la même lecture continuellement

Conclusion

Dans ce laboratoire, nous avons pu implémenter une interface qui peut lire des chaînes de caractères générées d'une manière fiable ou non-fiable. Nous avons pu tester cette interface en simulation et sur la carte. Les tests ont été effectués avec succès, et nous avons pu vérifier que l'interface fonctionne correctement.

Nous avons aussi fait contrôlé la partie 1 à Anthony Convers le 6. décembre et la partie 2 à M. Messerli le 13. décembre.

Code

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

/**
 * Address of the 32bit constant
 */
#define AXI_LW_CONST_REG            (AXI_LW_REG(0))

/* Base address for this PW's interface memory space */
#define AXI_LW_HPS_FPGA_LABO_ADD    0x010000

/* Gets the register address for this PW's memory space, with an offset */
#define AXI_HPS_LABO_REG(offset)    (AXI_LW_REG(AXI_LW_HPS_FPGA_LABO_ADD + offset))

/* Interface ID (RO) */
#define ARE_PW5_INTERFACE_ADDR      (0x00)
```

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               :
 * Date                 :
 *
 * Context              : ARE lab
 *
 ********************************************************************************************
 * Brief: Conception d'une interface évoluée sur le bus Avalon avec la carte DE1-SoC
 *
 ********************************************************************************************
 * Modifications :
 * Ver    Date        Student      Comments
 *
 *
********************************************************************************************/
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "axi_lw.h"
#include "io/io_function.h"
#include "gen/gen_function.h"

int __auto_semihosting;



/***************************
 * Macros
 **************************/

#define ARE_PW5_PRINT_BUFFER_SIZE   (256u)


/***************************
 * Typedefs & structures
 **************************/

/**
 * Function pointer called when a key is pressed
 */
typedef void(*TKeyPressHandler)();


/**
 * Structure containing both the action handler, and whether the key is pressed or not
 */
typedef struct
{
    /**
     * Handler when key is pressed
```

```
     * Does not fire again unless key has been released in the meantime
     */
    const TKeyPressHandler on_press;

    /**
     * Used to know whether the key has been released in the meantime or not
     */
    bool key_pressed;

    /**
     * Used to know whether continuous pressing is allowed
     */
    bool allows_continuous_pressing;

} TKeyPressAction;



/***************************
 * Global variables
 ***************************/

static size_t error_count = 0u;

static char print_buffer[ARE_PW5_PRINT_BUFFER_SIZE] = {0};
static int print_buffer_sp = 0u;


/***************************
 * Key pressed handlers
 ***************************/

void key0_press_handler(void)
{
    /* Init chars */
    AXI_HPS_LABO_REG(ARE_PW5_INIT_CHAR_ADDR) = ARE_PW5_INIT_CHAR_MASK;
} /* key0_press_handler */


void key1_press_handler(void)
{
    /* Only perform action when in manual mode */
    uint32_t mode = (AXI_HPS_LABO_REG(ARE_PW5_MODE_ADDR) & ARE_PW5_MODE_MASK) >> ARE_PW5_MODE_OFFSET
    if(ARE_PW5_MODE_MANUAL == mode)
    {
        AXI_HPS_LABO_REG(ARE_PW5_NEW_CHAR_ADDR) = ARE_PW5_NEW_CHAR_MASK;
    } /* if */
} /* key0_press_handler */


void key2_press_handler(void)
{
    char checksum_char = 0;
    char characters[ARE_PW5_CHARS_NUMBER] = {0};

    are_pw5_gen_read(characters, &checksum_char);
```

```
    // Verify checksum : (char_1 + char_2 + ... + char_16 + checksum) modulo 256 = 0
    int calcul_integrity = 0;
    for(size_t i = 0; i < ARE_PW5_CHARS_NUMBER; ++i)
    {
        calcul_integrity += characters[i];
    } /* if */
    calcul_integrity += checksum_char;
    calcul_integrity %= 256;
    bool checksum_ok = calcul_integrity == 0;

    int status = AXI_HPS_LABO_REG(ARE_PW5_STATUS_ADDR);

    print_buffer_sp += sprintf(print_buffer + print_buffer_sp,
                               "%s : status: %d , checksum: %d, calcul integrity: %s, string: %s\n",
                               checksum_ok ? "OK" : "ER",
                               status,
                               checksum_char,
                               checksum_ok ? "OK" : "ERROR",
                               characters);

    if(!checksum_ok)
    {
        // ER : nombre d'erreur cumulée : X
        ++error_count;

        print_buffer_sp += sprintf(print_buffer + print_buffer_sp,
                                   "ER : nombre d'erreur cumulée : %d\n",
                                   error_count);
    } /* if */

    printf("%s", print_buffer);

    /* Clear print buffer */
    memset(print_buffer, 0, print_buffer_sp);
    print_buffer_sp = 0;
} /* key0_press_handler */



/*************************
 * Main entrypoint
 *************************/

int main(void){

    TKeyPressAction key_press_actions[ARE_PW5_BUTTONS_NBR] = {
        {
            key0_press_handler,
            false,
            false
        },
        {
            key1_press_handler,
            false,
            false
        },
        {
            key2_press_handler,
```

```
        false,
        true
    }
};

/**
 * Previous ON / OFF states of the switches
 */
uint32_t prev_switches_state;

/**
 * Current ON / OFF states of the switches
 */
uint32_t curr_switches_state;

printf("Laboratoire: Conception d'une interface évoluée \n");
printf("Design standard ID: %08X\n", (unsigned int)AXI_LW_CONST_REG);
printf("Interface ID: %08X\n", (unsigned int)AXI_HPS_LABO_REG(ARE_PW5_INTERFACE_ADDR));

/*
 * Init I/O
 */
Keys_init();
Switchs_init();
Leds_init();

/*
 * Main program
 */

while(1)
{
    curr_switches_state = Switchs_read();

    /* Update LEDs according to the enabled switches */
    Leds_clear(ARE_PW5_LEDS_MASK);
    Leds_set(curr_switches_state);

    uint32_t switches_diff = curr_switches_state ^ prev_switches_state;
    if(switches_diff)
    {
        /* Check if we need to change the generation frequency */
        if(switches_diff & ARE_PW5_FREQ_SWITCHES_MASK)
        {
            are_pw5_gen_set_freq((curr_switches_state & ARE_PW5_FREQ_SWITCHES_MASK) >> ARE_PW5_F
        } /* if */

        /* Does the mode need to change? */
        if(switches_diff & ARE_PW5_MODE_SWITCHES_MASK)
        {
            are_pw5_gen_set_mode((curr_switches_state & ARE_PW5_MODE_SWITCHES_MASK) >> ARE_PW5_M
        } /* if */

        /* Are we toggling the reliable reading mode? */
        if(switches_diff & ARE_PW5_TRUSTY_SWITCHES_MASK)
        {
            are_pw5_gen_set_trusty(curr_switches_state & ARE_PW5_TRUSTY_SWITCHES_MASK);
        } /* if */
```

```
            prev_switches_state = curr_switches_state;
        } /* if */



        for(size_t i = 0; i < ARE_PW5_BUTTONS_NBR; ++i)
        {
            if(Key_read(i))
            {
                if(!key_press_actions[i].allows_continuous_pressing
                 && key_press_actions[i].key_pressed)
                {
                    continue;
                } /* if */

                key_press_actions[i].on_press();
                key_press_actions[i].key_pressed = true;
            }
            else
            {
                key_press_actions[i].key_pressed = false;
            } /* if */
        } /* for */
    } /* while */

    return EXIT_SUCCESS;
}
```

gen/gen_function.h

```
#ifndef SRC_GEN_GEN_FUNCTION_H_
#define SRC_GEN_GEN_FUNCTION_H_

#include <stdint.h>
#include <stdbool.h>


/**************************
 * Triggering switches
 **************************/

#define ARE_PW5_FREQ_SWITCHES_OFFSET    (8u)
#define ARE_PW5_FREQ_SWITCHES_MASK      ((0b11) << ARE_PW5_FREQ_SWITCHES_OFFSET)

#define ARE_PW5_MODE_SWITCHES_OFFSET    (7u)
#define ARE_PW5_MODE_SWITCHES_MASK      ((0b1) << ARE_PW5_MODE_SWITCHES_OFFSET)

#define ARE_PW5_TRUSTY_SWITCHES_OFFSET  (0u)    /* Part II */
#define ARE_PW5_TRUSTY_SWITCHES_MASK    (0b1)   /* Part II */


/**************************
 * Interface status (RO)
 **************************/
```

```
#define ARE_PW5_STATUS_ADDR             (0x10)
#define ARE_PW5_STATUS_OFFSET           (0x00)
#define ARE_PW5_STATUS_MASK             (0x00000003 << ARE_PW5_STATUS_OFFSET)
#define ARE_PW5_STATUS_TRUSTY           (0b10 << ARE_PW5_STATUS_OFFSET)
#define ARE_PW5_SNAPSHOT_AVAIL_MASK     (0b01 << ARE_PW5_STATUS_OFFSET)


/**************************
 * Modes & delays (RW)
 **************************/

#define ARE_PW5_DELAY_ADDR              (0x14)
#define ARE_PW5_DELAY_OFFSET            (0x00)
#define ARE_PW5_DELAY_MASK              (0x00000003 << ARE_PW5_DELAY_OFFSET)
#define ARE_PW5_DELAY_1HZ               (0b00 << ARE_PW5_DELAY_OFFSET)
#define ARE_PW5_DELAY_1KHZ              (0b01 << ARE_PW5_DELAY_OFFSET)
#define ARE_PW5_DELAY_100KHZ            (0b10 << ARE_PW5_DELAY_OFFSET)
#define ARE_PW5_DELAY_1MHZ              (0b11 << ARE_PW5_DELAY_OFFSET)

#define ARE_PW5_MODE_ADDR               ARE_PW5_DELAY_ADDR
#define ARE_PW5_MODE_OFFSET             (0x04)
#define ARE_PW5_MODE_MASK               (0x00000001 << ARE_PW5_MODE_OFFSET)
#define ARE_PW5_MODE_MANUAL             (0b0 << ARE_PW5_MODE_OFFSET)
#define ARE_PW5_MODE_AUTO               (0b1 << ARE_PW5_MODE_OFFSET)

#define ARE_PW5_TRUSTY_ADDR             (0x1C)
#define ARE_PW5_TRUSTY_OFFSET           (0x00)
#define ARE_PW5_TRUSTY_MASK             ARE_PW5_TRUSTY_SWITCHES_MASK


/**************************
 * Controls (WO)
 **************************/

#define ARE_PW5_INIT_CHAR_ADDR          ARE_PW5_STATUS_ADDR
#define ARE_PW5_INIT_CHAR_MASK          (1u)
#define ARE_PW5_NEW_CHAR_ADDR           ARE_PW5_STATUS_ADDR
#define ARE_PW5_NEW_CHAR_MASK           (1u << 4u)
#define ARE_PW5_STABLE_READ_REQ_ADDR    (0x18)
#define ARE_PW5_STABLE_READ_REQ_MASK    (0b1)


/**************************
 * Characters (RO)
 **************************/

#define ARE_PW5_CHAR_0_OFFSET           (0x00u)
#define ARE_PW5_CHAR_1_OFFSET           (0x01u)
#define ARE_PW5_CHAR_2_OFFSET           (0x02u)
#define ARE_PW5_CHAR_3_OFFSET           (0x03u)

#define ARE_PW5_CHARS_0_ADDR            (0x20)
#define ARE_PW5_CHARS_1_ADDR            (0x24)
#define ARE_PW5_CHARS_2_ADDR            (0x28)
#define ARE_PW5_CHARS_3_ADDR            (0x2C)

#define ARE_PW5_CHECKSUM_ADDR           (0x30)
#define ARE_PW5_CHECKSUM_OFFSET         (0x00)
```

```
#define ARE_PW5_CHECKSUM_MASK           (0xFF << ARE_PW5_CHECKSUM_OFFSET)

#define ARE_PW5_CHARS_BLOCKS            (4u)
#define ARE_PW5_CHARS_PER_ADDR          (4u)
#define ARE_PW5_CHARS_NUMBER            (16u)


/**************************
 * Functions
 **************************/

/**
 * Sets a new frequency for the char generator
 * \param new_frequency New frequency. Not in terms of Hz, but in terms of the values that we're giv
 */
void are_pw5_gen_set_freq
(
    uint32_t new_frequency
);

/**
 * Sets a new mode for the char generator
 * \param new_mode New mode
 */
void are_pw5_gen_set_mode
(
    uint32_t new_mode
);

/**
 * Reads the characters as sent by the board
 * \param enable When 1, enables that mode. Disables if 0
 */
void are_pw5_gen_read
(
    char  characters[ARE_PW5_CHARS_NUMBER],
    char* checksum
);

/**
 * Enables or disables the trustworthy reading mode (reliable reads)
 * \param enable When 1, enables the reliable reads. Disables if 0
 */
void are_pw5_gen_set_trusty
(
    char enabled
);


#endif /* SRC_GEN_GEN_FUNCTION_H_ */
```

gen/gen_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                 : io_function.h
 * Author               : Pedro Alves da Silva
 * Date                 : 29 november 2024
 *
 * Context              : ARE lab
 *
 ********************************************************************************************
 * Brief: I/O functions
 *
 ********************************************************************************************
 * Modifications :
 * Ver    Date        Student           Comments
 * 1.0    6 oct 24    pedro.alvesdas    First version
********************************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gen_function.h"
#include "../axi_lw.h"


void are_pw5_gen_set_freq
(
    uint32_t new_frequency
)
{
    int prev_value = AXI_HPS_LABO_REG(ARE_PW5_DELAY_ADDR) & ~ARE_PW5_DELAY_MASK;
    prev_value |= (new_frequency << ARE_PW5_DELAY_OFFSET) & ARE_PW5_DELAY_MASK;
    AXI_HPS_LABO_REG(ARE_PW5_DELAY_ADDR) = prev_value;
} /* are_pw5_gen_set_freq */


void are_pw5_gen_set_mode
(
    uint32_t new_mode
)
{
    /* Erase bits that about to be written */
    int prev_value = AXI_HPS_LABO_REG(ARE_PW5_MODE_ADDR) & ~ARE_PW5_MODE_MASK;
    prev_value |= (new_mode << ARE_PW5_MODE_OFFSET) & ARE_PW5_MODE_MASK;
    AXI_HPS_LABO_REG(ARE_PW5_MODE_ADDR) = prev_value;
} /* are_pw5_gen_set_mode */


void are_pw5_gen_read
(
    char  characters[ARE_PW5_CHARS_NUMBER],
    char* checksum
)
{
    /* Board is in a trusty mode (reads are guaranteed to be correct) */
    //printf("[DEBUG] Status: %X\n[DEBUG] Snapshot request reg: %X\n",
    //     (unsigned int)AXI_HPS_LABO_REG(ARE_PW5_STATUS_ADDR),
```

```
    //     (unsigned int)AXI_HPS_LABO_REG(ARE_PW5_STABLE_READ_REQ_ADDR));

    if(ARE_PW5_STATUS_TRUSTY == (AXI_HPS_LABO_REG(ARE_PW5_STATUS_ADDR) & 0x02))
    {
        //printf("[DEBUG] Trustworthy read mode\n");
        /* Request snapshot, and wait until it's available */
        AXI_HPS_LABO_REG(ARE_PW5_STABLE_READ_REQ_ADDR) = ARE_PW5_STABLE_READ_REQ_MASK;
        while(ARE_PW5_SNAPSHOT_AVAIL_MASK != (AXI_HPS_LABO_REG(ARE_PW5_STATUS_ADDR) & ARE_PW5_SNAPSH
        {
            //printf("[DEBUG] Waiting for snapshot\n");
            /* Wait */
        }
    } /* if */

    /* Read characters */
    const size_t base_char_offset = (ARE_PW5_CHARS_PER_ADDR - 1) * 8;
    for(size_t i = 0; i < ARE_PW5_CHARS_BLOCKS ; ++i)
    {
        uint32_t char_block = AXI_HPS_LABO_REG(ARE_PW5_CHARS_0_ADDR + i * sizeof(uint32_t));
        for(size_t j = 0; j < ARE_PW5_CHARS_PER_ADDR ; ++j)
        {
            characters[i * ARE_PW5_CHARS_PER_ADDR + j] = char_block >> (base_char_offset - j * 8) &
        } /* for */
    } /* for */

    *checksum = AXI_HPS_LABO_REG(ARE_PW5_CHECKSUM_ADDR);
} /* are_pw5_gen_read */

void are_pw5_gen_set_trusty
(
    char enabled
)
{
    int prev_value = AXI_HPS_LABO_REG(ARE_PW5_TRUSTY_ADDR);
    prev_value &= ~ARE_PW5_TRUSTY_MASK;
    prev_value |= (enabled << ARE_PW5_TRUSTY_OFFSET) & ARE_PW5_TRUSTY_MASK;
    AXI_HPS_LABO_REG(ARE_PW5_TRUSTY_ADDR) = prev_value;
} /* are_pw5_gen_set_trusty */
```

io/io_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                 : io_function.h
 * Author               : Pedro Alves da Silva
 * Date                 : 29 november 2024
 *
 * Context              : ARE lab
 *
 *****************************************************************************************
 * Brief: I/O functions
```

```
 *
 ********************************************************************************************
 * Modifications :
 * Ver    Date        Student           Comments
 * 1.0    6 oct 24    pedro.alvesdas    First version
********************************************************************************************/

#ifndef SRC_IO_IO_FUNCTION_H_
#define SRC_IO_IO_FUNCTION_H_

#include "../axi_lw.h"
#include <stdint.h>
#include <stdbool.h>


/***************************
 * I/O buttons (RO)
 ***************************/

#define ARE_PW5_BUTTONS_ADDR        (0x04)
#define ARE_PW5_BUTTONS_OFFSET      (0x00)
#define ARE_PW5_BUTTONS_MASK        (0x00000007)
#define ARE_PW5_BUTTONS_NBR         (3u)


/***************************
 * I/O switches (RO)
 ***************************/

#define ARE_PW5_SWITCHES_ADDR       (0x08)
#define ARE_PW5_SWITCHES_OFFSET     (0x00)
#define ARE_PW5_SWITCHES_MASK       (0x000003FF << ARE_PW5_SWITCHES_OFFSET)
#define ARE_PW5_SWITCHES_NBR        (10u)


/***************************
 * I/O LEDs (RO)
 ***************************/

#define ARE_PW5_LEDS_ADDR           (0x0C)
#define ARE_PW5_LEDS_OFFSET         (0x00)
#define ARE_PW5_LEDS_MASK           (0x000003FF << ARE_PW5_LEDS_OFFSET)
#define ARE_PW5_LEDS_NBR            (10u)


/***************************
 * Functions
 ***************************/

// Swicths_init function : Initialize all Switchs in PIO core (SW9 to SW0)
void Switchs_init(void);

// Leds_init function : Initialize all Leds in PIO core (LED9 to LED0)
void Leds_init(void);

// Keys_init function : Initialize all Keys in PIO core (KEY3 to KEY0)
void Keys_init(void);
```

```
// Switchs_read function : Read the switchs value
// Parameter : None
// Return : Value of all Switchs (SW9 to SW0)
uint32_t Switchs_read(void);

// 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_clear function : Clear to OFF some or all Leds (LED9 to LED0)
// Parameter : "maskleds"= Leds selected to apply a clear (maximum 0x3FF)
// Return : None
void Leds_clear(uint32_t maskleds);

// 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);

#endif /* SRC_IO_IO_FUNCTION_H_ */
```

io/io_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                 : io_function.h
 * Author               : Pedro Alves da Silva
 * Date                 : 29 november 2024
 *
 * Context              : ARE lab
 *
 *****************************************************************************************
 * Brief: I/O functions
 *
 *****************************************************************************************
 * Modifications :
 * Ver    Date        Student           Comments
 * 1.0    6 oct 24    pedro.alvesdas    First version
*****************************************************************************************/
```

```c
#include "io_function.h"

#include <stdio.h>
#include <stdbool.h>


static bool Keys_read(void)
{
    /* Keys are active low. Invert bits */
    int read_data = ~AXI_HPS_LABO_REG(ARE_PW5_BUTTONS_ADDR) & ARE_PW5_BUTTONS_MASK;

    /* Offset to align to LSb */
    read_data = read_data >> ARE_PW5_BUTTONS_OFFSET;

    return read_data ? true : false;
} /* Keys_read */


void Switchs_init(void)
{
    if(Switchs_read())
    {
        printf("%s", "Warning: some of the switches are in the ON position\n");
    } /* if */
}


void Leds_init(void)
{
    /* Reset all LED bits */
    Leds_write(0u);
}


void Keys_init(void)
{
    /* Keys are active low */
    if(Keys_read())
    {
        printf("%s", "Warning: some of the keys are being pressed\n");
    } /* if */
}


//*********************************//
//****** Global usage function ******//

uint32_t Switchs_read(void)
{
    int read_data = AXI_HPS_LABO_REG(ARE_PW5_SWITCHES_ADDR) & ARE_PW5_SWITCHES_MASK;
    return read_data >> ARE_PW5_SWITCHES_OFFSET;
} /* Switchs_read */


void Leds_write(uint32_t value)
{
    /* In this function, we either turn all LEDs ON or OFF */
    if(value == 0)
```

```
    {
        AXI_HPS_LABO_REG(ARE_PW5_LEDS_ADDR) &= ~AXI_HPS_LABO_REG(ARE_PW5_LEDS_ADDR);
    }
    else
    {
        Leds_set(ARE_PW5_LEDS_MASK);
    } /* if */
} /* Leds_write */


void Leds_set(uint32_t maskleds)
{
    /* Only enable the ones in the mask */
    AXI_HPS_LABO_REG(ARE_PW5_LEDS_ADDR) |= (maskleds << ARE_PW5_LEDS_OFFSET) & ARE_PW5_LEDS_MASK;
} /* Leds_set */


void Leds_clear(uint32_t maskleds)
{
    AXI_HPS_LABO_REG(ARE_PW5_LEDS_ADDR) &= (~maskleds << ARE_PW5_LEDS_OFFSET) & ARE_PW5_LEDS_MASK;
} /* Leds_clear */


void Leds_toggle(uint32_t maskleds)
{
    /* Only change required LEDs */
    AXI_HPS_LABO_REG(ARE_PW5_LEDS_ADDR) ^= (maskleds << ARE_PW5_LEDS_OFFSET) & ARE_PW5_LEDS_MASK;
} /* Leds_toggle */


bool Key_read(int key_number)
{
    bool ret_val = false;

    /* Only perform check if requested key number is valid */
    if(key_number >= 0 && key_number < ARE_PW5_BUTTONS_NBR)
    {
        /* Keys are active low. Invert bits */
        int read_data = ~AXI_HPS_LABO_REG(ARE_PW5_BUTTONS_ADDR) & ARE_PW5_BUTTONS_MASK;

        /* Offset to align to LSb, and apply key mask */
        read_data = read_data >> ARE_PW5_BUTTONS_OFFSET;
        read_data &= 1 << key_number;

        ret_val = read_data ? true : false;
    } /* if */

    return ret_val;
} /* Key_read */
```