BASYS Project Documentation

1 Overview

In this project we implemented a simulator for a SIMP processor, by programming Digilent's BasysMX3 card in C language.
This document holds descriptions for main functionalities in the project.

2 stopper.asm

This file holds the assembly program for the stopper functionality.
The program displays the time that passed since the last Reset.
The main loop checks the value of `IORegister[0]` which goes up by 1 each 32 msecs, if there is a predefined difference since the last time the register was tested for it's value, the stopper goes up by 1 second.
It also tests the values for `IORegister[2]` and `IORegister[3]` which hold counts for number of pressings on BTNC and BTND respectively.

3 Timers

Timer interrupts are used in the program in order to time the execution of the assembly commands and sample the buttons in order to recognize presses without blocking the execution of the program while doing so.

3.1 Timer1

`Timer1` is used to generate interrupts every approximately 3 ms. Every time the interrupt handler routine is called, the following operations are performed

- If the interrupt was called 15 times, 32 ms elapsed, then IORegister[0] is updated and a single command is executed
- The SSD is refreshed

3.2 Timer5

`Timer5` is used to sample the buttons state without blocking the program and it generates an interrupt every approximately 300 micro sec.
Every time the interrupt handler routine is called, we swipe through all 5 buttons using `BTN_GetValue()`.

For `BTNU`, `BTNL` and `BTNR` we also hold their previous state, i.e their state at the previous interrupt.

```
buttonValue = BTN_GetValue(BUTTON_RIGHT);
if (!buttonValue && !btnState.BTNR)
{
    btnState.BTNR = currentButtonState.BTNR ;
}
```

Simply put: we define the state of the button to be ‘1’ if and only if the button was pressed and then released, making the button handler to

ignore continuous long pressings.

For `BTND` and `BTNC` we use specific functions for updating the `IORegisters` array.

4 commons.h

The `commons.h` header file groups all structs and variables which are used by all source files in the program.

4.1 Structs

- `ExecutionState` - holds all relevant fields for the execution of the program
- `SwtState` - holds the state of the switches
- `BtnState` - holds the state of the buttons
- `Instruction` - represents a decoded MIPS assembly instruction

4.2 Fields

- `registers` - an array holding the state of the registers
- `memory` - an array representing the memory of the system
- `executionState` - holds the current execution state of the program
- `btnState` - holds the current state of the buttons
- `decodedInstruction` - holds the current instruction being executed
- `instructionCounter` - holds the number of instructions

executed

- `swtState` - holds the current state of the switches

5 simulator

The `simulator` source file groups the functions that control the execution of the simulation.

5.1 Fields

- `stopperInputMemory` an array holding the input memory of the stopper program
- `fibonachiInputMemory` an array holding the input memory of the Fibonachi program

5.2 Functions

initSimulator

```
void initSimulator()
```

Description

This function initialized the fields that are used for running a simulation. It loads the memory of the program to be simulated according to the state of `SW7`.

execute

```
void execute();
```

Description

This function executes a single step of the simulation. It sets the display according the switch states and then executes a single command from the memory unless the simulation is paused because `BTNL` was pressed. If `BTNR` was pressed while the simulation is paused, a single command is executed.
The simulation continues to run until a `HALT` command is executed.

6 lcd_handler

The `lcd_handler` source file groups all functions that are responsible for setting the LCD according to the switch states.

6.1 Fields

- `registerNumber` - the register number to be presented on screen
- `currentSwitch12Case` - the state of switches 0,1 in order to change the lcd display adequately

6.2 Functions

initLcdHandler

```
void initLcdHandler();
```

Description

Initializes the global parameters.

getLcdState

```
void getLcdState();
```

Description

Called from `execute()` in simulator file in every iteration.
Determines according to global `currentSwitch12Case` the needed case for action and calling to one of the needed functions in every iteration of the program:

All methods except `lcdShowInstructionandPc` use `BTNU` to receive additional state changes.

1. `void lcdShowInstructionandPc()`

   Displays the instruction number and the pc counter while the assembly program is running.

2. `void lcdShowSelectedRegister()`

   Displays the register number and its value.

3. `void lcdShowSelectedMemory()`

   Displays selected memory adress and it’s value and the pc counter below.
   Uses switches 5,6 to state the initial mem address.

4. `void lcdShowInstructionCounter()`

   Displays the total amount of instructions were operated so far.

clearLcdDisplay

```
void clearLcdDisplay();
```

Description

Clears lcd display before any needed display change, in order to display the information properly.

7 button_state_handler

The `button_state_handler` source file groups all functions that handle the sampling of button states. It uses `Timer5` in order to raise the flags

without blocking the program.

Most of the functionality is described under the `Timers` chapter.

7.1 Fields

- `currentButtonState` - a struct of type `BtnState` which holds the current state of the buttons.

7.2 Functions

buttonStateHandlerInit

```
void buttonStateHandlerInit();
```

Description

Initializes all needed values of button states and sets the timer.

Timer5Handler

```
void __ISR(_TIMER_5_VECTOR, ipl2) _Timer5Handler(void);
```

Description

This is the handler function for the timer.

It’s functionality is described under the `Timers` section.

8 io_registers_handler

The `io_registers_handler` source file groups functions which update the IO registers which can be used by the assembly program.

8.1 Fields

- `IORegisters` - an array containing the values of the IO registers in the program.

8.2 Functions

setIORegister

```
void setIORegister(int registerIndex, int value);
```

Description

Sets the value of the IORegister specified by the `registerIndex` parameter to the value specified by the `value` parameter.
This function does not change values for registers that are defined as Read Only.

updateCounterRegister

```
int updateCounterRegister();
```

Description

Updates the value of IORegister[0] by 1.

This functions is called every 32 ms by `Timer1` interrupt.

updateButtonXRegister

```
void updateButtonDownRegister();
void updateButtonCenterRegister();
```

Description

Updates the value of the proper IORegister by 1.

This function is called by the `Timer5` handler each time a press is detected.

9 instruction_executer

The `instruction_executer` source file groups all functions responsible for executing a single assembly instruction.

9.1 Functions

executeInstruction

```
void executeInstruction(
    Instruction* instruction,
    int memory[],
```

```
    int registers[],
    ExecutionState* state);
```

Description

This function handles the execution of an instruction according to the instructions opcode. It delegates the execution to helper functions within the source file.