so, the XADC module should also be used in connection with the joystick.

13.11 Translator

We can design a digital system to translate voice commands from English to Spanish (or another language) and show them on a 16×2 LCD. The system will have two parts. The first part will recognize the spelled out English word. We can use the EasyVR shield for this purpose [60]. This module has predefined speaker-independent word sets. Also, you can create your own speaker-dependent word set. In the second part of the translator system, we will get the recognition result and form a state machine in the FPGA to provide the translated word corresponding to the recognized one. Then, this word is shown on the LCD.

The LCD we have used in our system is WH1602N with a built-in controller ST7066 or equivalent. To use the LCD display, we need a Verilog description. We provide such an LCD driver module in Listing 13.16. This module has four inputs: `clk`, `reset`, `wr_en`, and `data_in`. The main clock signal, `clk`, is expected to be 100 MHz. Active-high `reset` signal resets the operation and the module waits for an active-high write enable `wr_en` signal. `data_in` is eight-bit data that will be transmitted to the display. The module has three outputs: `data_out` (eight-bit data output), `en` (enable signal to drive the LCD), and `rs` (data/instruction selection signal for LCD).

Listing 13.16 Verilog Description of LCD Driver Module

```
module LCD_driver(clk,reset,wr_en,data_in,data_out,en,rs);

input clk;
input reset;
input wr_en;
input [7:0] data_in;
output reg [7:0] data_out;
output reg en;
output reg rs;

parameter clk_param = 100000;

localparam INIT=2'b00,WAIT=2'b01,WRITE=2'b10;
reg [2:0] state = INIT;

localparam init_index=3;
localparam char_index=15;
reg [3:0] init_count=0;
integer limit_count=0;
reg [7:0] init [3:0];
reg [3:0] clear=0;

initial begin
init[0]=8'h30; // 1 Line, 5x8 Dots
init[1]=8'h01; // Clear display
init[2]=8'h06; // Increment cursor (Shift cursor to right)
init[3]=8'h0F; // Display on cursor blinking
rs=1'b1;
end

always@(negedge clk)
begin
rs <= 1'b1;
en <= 1'b1;
if (reset)
```

```
    begin
    state <= INIT;
    init_count <= 0;
    limit_count <= 0;
    clear <= 0;
    end
else
    begin
    case (state)
    INIT:
    begin
    rs <= 0;
    data_out <= init[init_count];
    if (limit_count == clk_param)
        begin
        en <= 0;
        limit_count <= 0;
        init_count <= init_count + 1'b1;
        if (init_count == init_index)
            begin
            init_count <= 0;
            state <= WAIT;
            end
        end
    else
        limit_count <= limit_count + 1;
    end
    WAIT:
    if (wr_en)
        state <= WRITE;
    WRITE :
    begin
    data_out <= data_in;
    if (limit_count == clk_param)
        begin
        en <= 0;
        limit_count <= 0;
        clear <= clear + 1'b1;
        if (clear == char_index)
            begin
            state <= INIT;
            clear <= 0;
            end
        else
            state <= WAIT;
        end
    else
        limit_count <= limit_count + 1;
    end
    endcase
    end
end
endmodule
```

The working principle of the LCD driver module (as a state machine) is as follows. The module starts in `INIT` state where it initializes the display by the predefined eightbit commands. Using them, we set the display for one line and 5 × 8 dots; then cleared the display screen; set the cursor direction; and changed the cursor to blinking mode. After initialization, the machine goes to `WAIT` state where it waits for `wr_en` signal to go to logic level 1. Once this happens, the machine goes to `WRITE` state. Here, it transfers `data_in` to `data_out` and waits for 1 microsecond. Afterward, the machine turns back to `WAIT` state by incrementing `clear` vector by one and waits for another `wr_en` signal. Once `clear` reaches 15 in decimal form, that means it reached the end of the line and it turns back to `INIT` state and clears the display.

We can connect the LCD to the Basys3 board as follows. The LCD's eight-bit data bus line (`DB0` to `DB7`) should be connected to JA port of the board (starting from JA[0] to JA[7]). Enable signal of the LCD (`E`) should be connected to JB[0]. Similarly, `RS` signal of the LCD should be connected to JB[2]. You can connect the `R/W` port of the LCD to ground since we will always be in write mode. Also, do not forget to supply V*DD* port of LCD with 5 V and connect V*SS* to ground. There is a contrast port V*O* on the LCD. This port can be connected to ground for maximum contrast. Finally, `A` and `K` ports control the back light of the LCD screen. You can set `A` to 5 V or 3.3 V, and `K` goes to the ground to lit your LCD screen.

The EasyVR shield communicates through the UART interface. Hence, we can use the UART transmitter and receiver blocks introduced in Chap. 12. After activating the EasyVR shield, it recognizes words in its predefined word set 1 as default. This word set includes English words `Action`, `Move`, `Turn`, `Run`, `Look`, `Attack`, `Stop`, and `Hello`. The Spanish translation of these words are `Accion`, `Movimiento`, `Giro`, `Correr`, `Mirar`, `Ataque`, `Detener`, and `Hola`, respectively.

Assuming that the reader does not have an EasyVR module, we simulate the translation operation by feeding input signals via the first eight switches of the Basys3 board. We provide the Verilog description of the top module for the translator constructed this way in Listing 13.17. This module has three inputs: `clk` (main clock of Basys3), `reset` (active-high reset signal), and `sw` vector (first eight switches on the Basys3 board). The outputs of the module are `rs`, `en`, and `data_out`, all of which are LCD driving signals.

Listing 13.17 Translator Implemented on the Basys3 Board in Verilog

```
module translator_topmodule(clk,reset,sw,rs,en,data_out);

input clk;
input reset;
input [7:0] sw;
output rs;
output  en;
output [7:0] data_out;

parameter clk_param =16000000;

reg [7:0] data [15:0];
reg [7:0] character;
wire [7:0] data_in;
integer counter=0;
reg [3:0] index=0;
reg wr_en=0;

LCD_driver_0 lcd1(.clk(clk),.reset(reset),.wr_en(wr_en),.data_in
    (data_in),
.data_out(data_out),.en(en),.rs(rs));

assign data_in = character;

always @ (posedge clk)
begin
if (reset)
    counter <= 0;
else
    begin
    if (counter == clk_param)
        counter <= 0;
    else
        counter <= counter + 1;
    end
end

always @ (posedge clk)
begin
if (reset)
    begin
    wr_en <= 0;
    index <= 0;
    end
else
    begin
    if (counter == clk_param)
        begin
        wr_en <= 1'b1;
        character <= data[index];
        index <= index + 1'b1;
        end
    else
        wr_en <= 0;
    end
end

always @ (posedge clk)
case (sw)
8'h01:
begin // ACTION - ACCION
data[0]<="A"; data[1]<="C"; data[2]<="C"; data[3]<="I";
data[4]<="O"; data[5]<="N"; data[6]<=" "; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h02:
begin // MOVE - MOVIMIENTO
data[0]<="M"; data[1]<="O"; data[2]<="V"; data[3]<="I";
data[4]<="M"; data[5]<="I"; data[6]<="E"; data[7]<="N";
data[8]<="T"; data[9]<="O"; data[10]<=" "; data[11]<=" ";
```

```
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h04:
begin // TURN - GIRO
data[0]<="G"; data[1]<="I"; data[2]<="R"; data[3]<="O";
data[4]<=" "; data[5]<=" "; data[6]<=" "; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h08:
begin // RUN - CORRER
data[0]<="C"; data[1]<="O"; data[2]<="R"; data[3]<="R";
data[4]<="E"; data[5]<="R"; data[6]<=" "; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h10:
begin // LOOK - MIRAR
data[0]<="M"; data[1]<="I"; data[2]<="R"; data[3]<="A";
data[4]<="R"; data[5]<=" "; data[6]<=" "; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h20:
begin // ATTACK - ATAQUE
data[0]<="A"; data[1]<="T"; data[2]<="A"; data[3]<="Q";
data[4]<="U"; data[5]<="E"; data[6]<=" "; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h40:
begin // STOP - DETENER
data[0]<="D"; data[1]<="E"; data[2]<="T"; data[3]<="E";
data[4]<="N"; data[5]<="E"; data[6]<="R"; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
8'h80:
begin // HELLO - HOLA
data[0]<="H"; data[1]<="O"; data[2]<="L"; data[3]<="A";
data[4]<=" "; data[5]<=" "; data[6]<=" "; data[7]<=" ";
data[8]<=" "; data[9]<=" "; data[10]<=" "; data[11]<=" ";
data[12]<=" "; data[13]<=" "; data[14]<=" "; data[15]<=" ";
end
default:
begin // MAKE A SELECTION - HAS UNA ELECCION
data[0]<="H"; data[1]<="A"; data[2]<="S"; data[3]<=" ";
data[4]<="U"; data[5]<="N"; data[6]<="A"; data[7]<=" ";
data[8]<="E"; data[9]<="L"; data[10]<="E"; data[11]<="C";
data[12]<="C"; data[13]<="I"; data[14]<="O"; data[15]<="N";
end
endcase

endmodule
```

The top module in Listing 13.17 uses the `LCD_driver` module to show the translation results. The top module has an internal `counter` which counts up to 160 milli-seconds. If `reset` signal goes to logic level 1, then `counter`, `wr_en`, and `index` values will be equal to logic level 0. When `counter` reaches `clk_param` (corresponding to 160 milliseconds), index of `data` memory is loaded into eight-bit `character` vector. This is directly connected to `data_in` of the `LCD_driver` module. There is a case statement at the end of the top module which loads Spanish translation corresponding to the given command (or English word). For our application, depending on which switch is at logic level 1, the corresponding word is loaded to `data` memory. Hence, each character of this word is displayed with the help of the `LCD_driver` module. The reader can modify this section if translation to another language is desired.

13.12 Air Freshener Dispenser

We can modify the air freshener dispenser system developed for the MSP430 microcontroller to work on the Basys3 board [32]. The system will have four different programs to spray fresh odor in 5-, 10-, 15-, and 20-second intervals. These values should be in minutes in an actual system. However, we set such values to observe the system output. The system should have a counter for these operations. When counter reaches the designated time value, the kit sprays the fresh odor and restarts counting again. We can use two switches to select among four programs. Besides, there should be an instant spray button. When it is pressed, the fresh odor should be sprayed and the counter should be reset. When the user selects another program, the counter should restart again. There should also be an on/off switch for the system. Spraying fresh odor can be indicated by blinking an LED on the board for three seconds.

13.13 Obstacle-Avoiding Tank

We can modify the obstacle-avoiding tank system developed for the MSP430 microcontroller to work on the Basys3 board [32]. Hence, we will build a tank which is driven by two stepper motors. The proximity sensors on the front edges of the tank will be used to sense obstacles on the way. The tank will change its direction by controlling motor speeds accordingly. The proximity sensor we have used in previous applications can also be employed for this application. By using the tuning screw on the sensor, the designer can adjust the distance the tank will turn when it faces an obstacle.

The sensors can be connected to JB or JC ports of the Basys3 board. The motor driver should also be connected to the JA port of the board. Since this application will be integrated on a tank, the board itself can be powered by a battery to ensure autonomy of the tank. Hence, 5 V has to be applied to external power pins of the board. If the battery's voltage is above 5 V, a regulator should be used.

13.14 Intelligent Washing Machine