Proj1Demo Example Design Description

Proj1Demo provides a partial Verilog HDL framework that must be extended to get the demo running. The design example also includes a PicoBlaze™ assembly language program (the "firmware") that demonstrates the basic functions of BotSim. The Proj1Demo framework can be extended to complete Project 1.

User Interface

The Proj1Demo firmware implements a modified version of the Getting Started project.

PUSHBUTTONS

The pushbutton switches control two wheel motors and reset the system as shown:

Pushbutton	Motor Function
BTN_LEFT	Left Forward
BTN_UP	Left Reverse
BTN_RIGHT	Right Forward
BTN_DOWN	Right Reverse
BTN_CENTER	System Reset

If neither of the two buttons that control a motor are pushed then that motor is stopped. If *both* of the buttons that control a motor are pushed the actions cancel each other leaving that motor stopped. If all 4 buttons are pushed at the same time the actions all cancel each other leaving both motors stopped.

LEDS

The eight LEDs on the Nexys 3 board are driven with the current value of the BotSim *Sensors* register. The contents of the *Sensors* register are described in the *BotSim Functional Spec.*

SEVEN SEGMENT DISPLAY

The 4 digits of the display are used to display *either* the current location of the RojoBot in its virtual world, *or* the direction of movement and compass heading indications, The display mode depends on the position of the rightmost slide switch SW[0].

Mode 1: SW[0] OFF (LO)

Digits 3 (leftmost) and 2 – Hexadecimal display of the X coordinate of the RojoBot's current location

Digits 1 and 0 (rightmost) – Hexadecimal display of the Y coordinate of the RojoBot's current location

Decimal point 0 (rightmost) – Blinks when the RojoBot stops because it is about to run into a wall.

Mode 2: SW[0] ON (HI)

Digit 3 (leftmost) – Displays the current RojoBot movement using the additional character codes provided in the seven segment display module. The following character mapping is used to show the movement:

Movement	**BotInfo Register Value**	**Display Character**	**Character Code Decimal (Hex)**
Halted (stopped)	00	H (upper case H)	24 (0x18)
Forward	04	F (upper case F)	15 (0x0F)
Reverse	08	b (lower case B)	11 (0x0B)
Slow left turn (1X)	0C	L (upper case L)	25 (0x19)
Fast left turn (2X)	0D	l (lower case L)	27 (0x1B)
Slow right turn (1X)	0E	R (upper case R)	26 (0x1A)
Fast right turn (2X)	0F	r (lower case R)	28 (0x1C)

Digits 2, 1, and 0 (rightmost) – Decimal display of the current orientation (compass heading) in degrees as specified in Project 1.

Decimal point 0 (rightmost) – Blinks when the RojoBot stops because it is about to run into a wall.

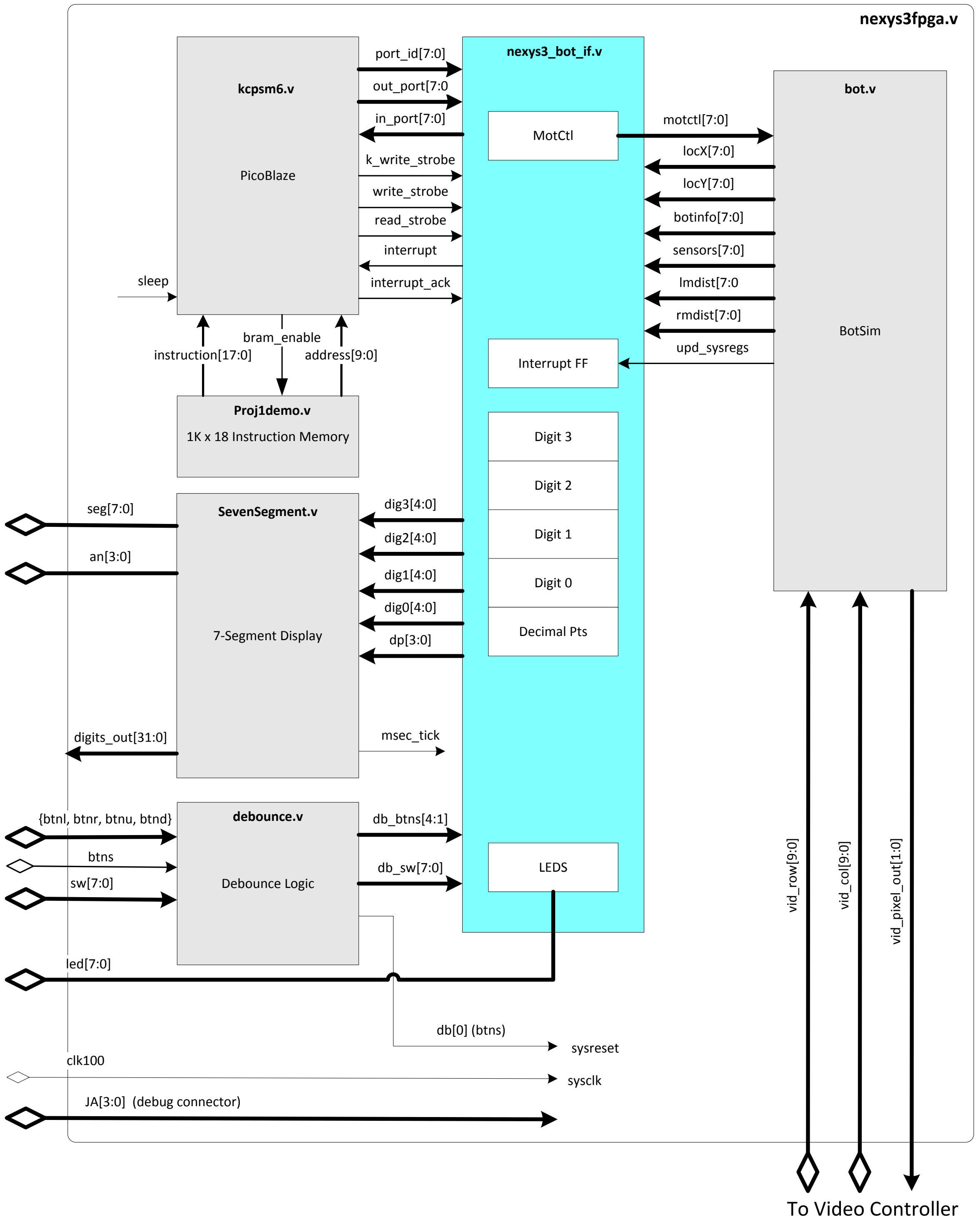
SLIDE SWITCHES

Slide SW[7:1] – These switches are not used

Slide SW[0] – Mode switch for the display. When SW[0] is OFF the RojoBot's current location is displayed in Hex. When SW[0] is ON the RojoBot's motion indicator and heading are displayed.

Hardware

The block diagram below shows the design hierarchy for the demo:



`nexys3fpga.v`
You have to create this module to provide the top level module for the example design. *nexys3fpga.v* instantiates the modules that make up the design. Don't forget to include *nexys3fpga.ucf* (the constraints file) in your project. You should be able to use the same file as was used in the Getting Started project with litte or no changes.

The modules to be included in the to level are described below:

`bot.v`
bot.v is the top level module for the *BotSim* RojoBot simulator. The register interface to the BotSim is described in the *BotSim 2.0 Functional Specification*. The internal hardware details and functions of the assembly language program that runs on the PicoBlaze CPU embedded within BotSim, are described in the *BotSim 2.0 Theory of Operation.* It isn't necessary to read the Theory of Operation document to get the demo working (or even to complete the project) but you may find it interesting.

`debounce.v`
debounce.v conditions the inputs signal from the pushbuttons and switches on the Nexys 3 board. It is identical to `debounce.v` used in the Getting Started project.

`kcpsm6.v`
kcpsm6.v is the PicoBlaze module. It is instantiated twice in the demo: Once as part of BotSim and again in *nexys3fpga.v* as the CPU that the demo application runs on. The latest version of the PicoBlaze, along with a set of development tools, users guides and some application notes can be downloaded from the Xilinx web site.

`Proj1demo.v`
Proj1demo.v is the program memory that contains the application for Proj1Demo. The program is written in PicoBlaze Assembly Language and created by the kcpsm6 assembler (*kcpsm6.exe*).

`nexys3_bot_if.v`
You have to design and implement this module to provide the interface between the PicoBlaze and *bot.v*. Its ports connect to the PicoBlaze CPU's port_id, input and output buses, and read and write strobes. The module also receives the debounced pushbutton and switch inputs from `debounce.v` and creates a register-based interface to the LED's and the 7-segment display on the Nexys3. The module should also include the interrupt flip-flop. The circuitry used to interface a PicoBlaze to external peripherals is described in the *KCPSM6 User Guide*.

Hint 1: *You build the I/O port map from Proj1Demo.psm. The port addresses are listed as constants (ex:* CONSTANT PA_MOTCTL_IN sets the port address of the BotSim motor control inputs to 0x09*) in the program.*

Hint 2*:* You may find the file *`kcpsm6_design_template.v`* useful in providing examples of Picoblaze instantiation, general purpose I/O, and the interrupt circuitry.

`SevenSegment.v`
This version of the seven segment display interface is identical to the one used in the Getting Started project.

Software

Proj1Demo.psm contains the assembly language program for the demo. The majority of the Project 1 program that you need to write is identical to this application.

NOTE: PROJ1DEMO ASSUMES THE PUSHBUTTONS INPUT HAS THE FOLLOWING BIT ORDERING:

BIT[7:5] :	RESERVED
BIT[4]:	BTN_CENTER (FORCES A HARDWARE SYSTEM RESET)
BIT[3]:	BTN_LEFT
BIT[2]:	BTN_UP
BIT[1]:	BTN_RIGHT
BIT[0]:	BTN_DOWN

INTERRUPT SERVICE ROUTINE – ISR()

The BotSim generates a signal called *upd_sysregs* that can be used as an interrupt source for the PicoBlaze. *upd_sysregs* pulses ($0 \rightarrow 1 \rightarrow 0$) every 50ms whether or not the BotSim state has changed. The function of this interrupt service routine (also called an interrupt handler) is to read new values from the BotSim and then "wake up" the main program so that it can update the Rojobot's state, the display and the LEDs. This is done in *proj1demo.psm* by using a flag called a semaphore (SP_SEM in this program). The flag is incremented by the interrupt handler just before it finishes its processing. The main program does a "busy-wait" loop on the flag waiting for it to become something other than 0. When it does, the main program updates the display and LEDs, does whatever other processing it needs to do, decrements the flag and jumps back to the busy-wait loop. The interrupt handler and display are kept in sync this way.

MAIN LOOP

Like most embedded programs, the demo program is an infinite loop. After initializing the tables and variables the program enters the main loop (label *main_L0*) which starts with a busy-wait loop that tests the SP_SEM flag.

When the new values are available (SP_SEM > 0) the program reads the switches to determine the display mode. If bit 0 (SW[0]) is '1' the program gets the hex values of the X and Y coordinates, converts the digits into 7-segments values by calling *hex2ssg()* and places the results in PicoBlaze registers Dig3, Dig2, Dig1, and Dig0. If SW[0] is '0' the program uses the BotInfo register to update the compass heading (*next_hdg()*) and the movement (*nxt_mvmt()*). Those functions also place their results in Dig3, Dig2, Dig1, Dig0. Once Dig3, Dig2, Dig1, and Dig0 have been updated the loop continues (label *main_L2*) by writing the new digits to the 7-segment display, writing the Sensor register to the LEDs, and getting the next step for the RojoBot (*next_step()*) by reading the pushbuttons. The last thing done in the main loop before jumping back to the busy-wait loop is to decrement the SP_SEM flag so that the interrupt handler knows that the display has been updated.

MOVEMENT DIRECTION TO CHARACTER CODE CONVERSION – MVMT2CC()

The *init_mvmttbl ()* and *mvmt2cc ()* functions provide an example for how to use the PicoBlaze scratchpad RAM and FETCH and STORE register indirect instructions to implement a lookup table. The lookup table for this function uses the 4-bit movement code from BotInfo to index into a table that contains the character code to display.

Since the scratchpad RAM is initialized to all zeros there must be a way to load the lookup table values into the scratchpad RAM. This is done in the *init_mvmttbl ()* function. *init_mvmttbl ()* loads register s0 with the base address of the lookup table. It then loads constants (SP_MVMT0, SP_MVMT1 …) into sequential table locations in the scratchpad RAM. *mvmt2cc ()* expects the value to convert in register s0 and does the lookup by simply adding that value to the base address of the lookup able and using a FETCH register indirect instruction to read the corresponding value from the table. The character code is returned in s1.

CALCULATING THE NEW HEADING – NEXT_HDG()

The *next_hdg()* function is used to translate the RojoBot's orientation into degrees. This could have been done in a lookup table containing 3 decimal numbers corresponding to the heading but doing that would have consumed 16-bytes of the scratchpad RAM. Proj1Demo uses a case statement to do the translation. The *next_hdg()* function gives an example of how to implement a C **switch** (case) statement in PicoBlaze Assembly Language. The source code is documented, but the short explanation is that each case consists of a compare/jump combination which compares the switch statement value with one of the valid cases and jumps to the next case if the compare fails. The **break** statement for each case is done with a jump to the end of the switch code.

NEXYS3 I/O MODULES API (APPLICATION PROGRAM INTERFACE)

The LEDs, debounced pushbuttons and switches and the seven segment display digits and decimal points are accessed by using short functions which, to some degree, isolate the hardware specifics from your application. The demo program uses the following functions:

DEB_rdbtns()	DEB_rdsw()	LED_wrleds()
SS_wrdigx()	SS_wrdpts()	

Examine the source code comments for calling conventions, usage, and functionality.