Proj2Demo Example Design Description
(Last updated 17-Oct-2016)

Proj2Demo 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 Proj2Demo framework can be extended to complete Project 2.

User Interface

The Proj2Demo firmware implements a modified version of Project 1 (this time done in software).

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	Not Used
BTN_CPU_RESET	System Reset

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 they cancel each other leaving both motors stopped.

LEDS

The low order (rightmost) eight LEDs on the Nexys4 DDR 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.* The high order eight LEDs are unused by the demo.

SEVEN SEGMENT DISPLAY

Digits[7:5] (the leftmost) of the display show the current heading (0 degrees, 45 degrees, etc.) of the Bot.

Digit[4] of the display shows the current movement (**F**orward, **b**ackwards, slow **L**eft turn, slow **R**ight turn, fast **l**eft turn, fast **r**ight turn, or **H**alted).

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[3:2] of the display show the current X coordinate (column) of the Rojobot. The output is in Hex.

Digits[1:0] of the display show the current Y coordinate (row) of the Rojobot. The output is in Hex.

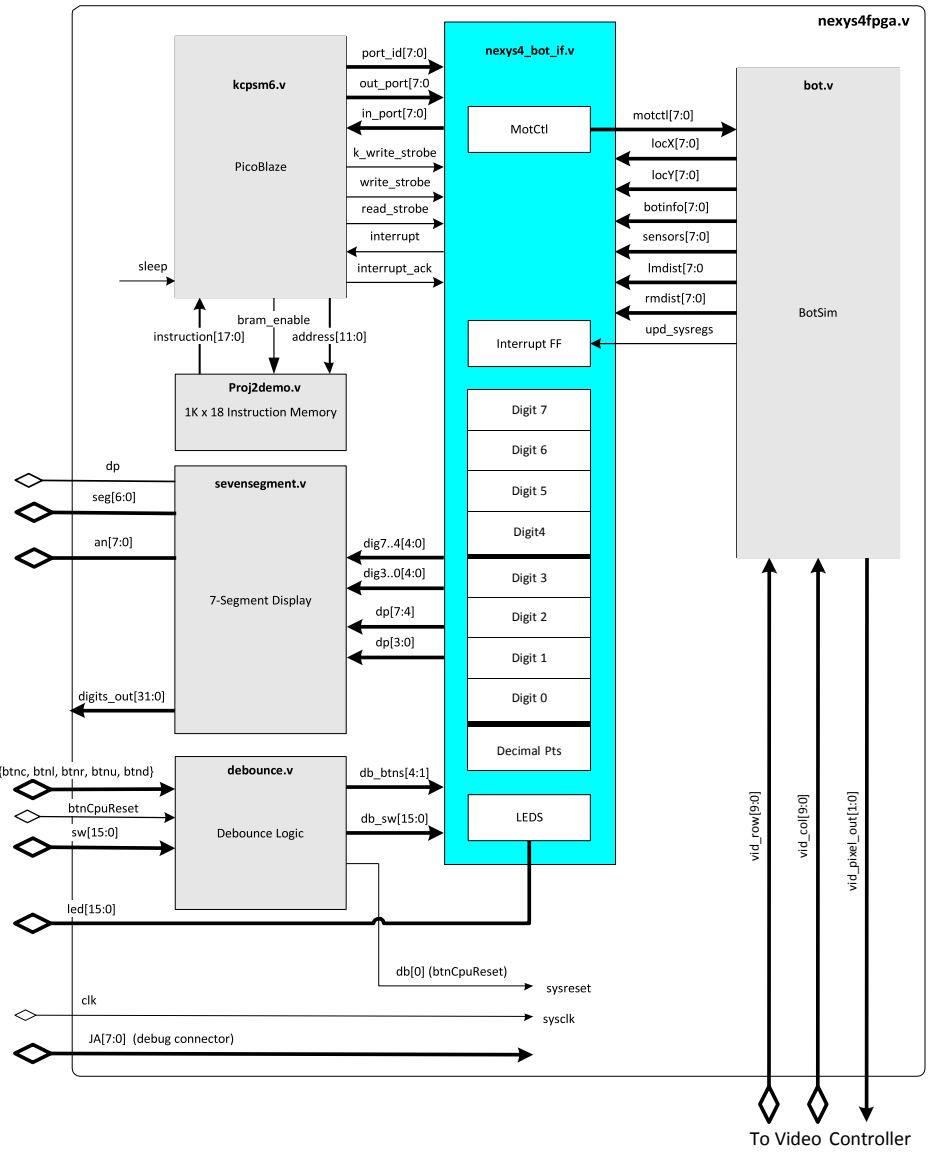
Decimal point 0 (rightmost) – Blinks in response to the *upd_sysregs* signal. You should observe the decimal point blinking rapidly.

SLIDE SWITCHES

The switches are not used in this demo.

Hardware

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



`nexys4fpga.v`

You have to create this module to provide the top level module for the example design. *nexys4fpga.v* instantiates the modules that make up the design. Don't forget to include the `n4DDRfpga_novideo.xdc` constraints file in your project to make sure your top level ports are mapped to the correct pins.

The modules to be included in the project 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 Nexys4DDR board.

kcpsm6.v
kcpsm6.v is the PicoBlaze module. It is instantiated twice in the demo: Once as part of BotSim and again in *nexys4fpga.v* as the CPU that the demo application runs on.

Proj2demo.v
Proj2demo.v is the program memory that contains the demo application. The program is written in PicoBlaze Assembly Language and created by the kcpsm6 assembler (*kcpsm6.exe*).

nexys4_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 Nexys4. 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 Proj2Demo.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
Drives the eight 7-segment display digits and decimal points on the Nexys4.

Software

Proj2Demo.psm contains the assembly language program for the demo. The majority of the Project 2 program that you need to write is identical to this application.

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

- BIT[7:5] : RESERVED
- BIT[4]: BTN_CENTER
- 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→1→0) every periodically 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 loop in the program so that it can update the Rojobot's state, the display and the LEDs.

"Waking up" the main loop is done 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 main loop calls the functions that update the program state (*next_loc()*. *nxt_mvmt(),* and *next_hdg()*). Those functions also place their results in Dig 7, Dig6, Dig5 Dig4, Dig3, Dig2, Dig1, and Dig0. Dig4, Dig2, Dig1, and Dig0 are stored in some of the global registers. Dig7, Dig6, Dig5, and Dig4 are stored in the scratchpad RAM because there weren't enough general purpose registers available. As assembly language programmers on a resource-constrained CPU like the PicoBlaze you will make these tradeoffs frequently. 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. The demo program 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.

NEXYS4 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()	DEB_rdws_hi()	LED_wrleds()
LED_wrleds_hi()	SS_wrdigx()	SS_wrdigx_hi()	SS_wrdpts()
SS_srdpts_hi()			

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