Robot Motion Theory of Operation

by

Michael Bourquin & Jean Shirimpaka

I. Introduction

In this project, we use the Diligent Nexys4 DDR development board to model a SimpleBot. The SimpleBot, a "virtual" robot, is a two-wheel platform with each driven by an independent motor while the third wheel stabilizes the robot. We utilize push buttons, switches and the seven-segment display on the Nexys4 board to control the virtual robot's wheel motors and display information about the robot's motion. In addition, we add a Verilog code to implement a seven-segment display peripheral by connecting it to the AHB-Lite bus so that the display digits can be written by a program running on the MIPSfpga core.

II. Method

Design of this project is consists of integrating both hardware and software, respectively described in section II.1 and section II.2 below. Hardware development consists of developing a seven-segment display peripheral module in Verilog to decode and drive the correct pin outputs to the display. This module is mapped via address select signals using the AHB-Lite bus in order to communicate with the MIPSfpga processor core, which drives the data inputs to the display. Software development consists of designing and implementing the logic which controls both the compass and the motion display and writes out to the hardware using MIPS assembly processor commands.

II.1 Hardware Development

An AHB-Lite peripheral for the MIPSfpga, in Fig.1, has been implemented to drive the seven-segment display of the Nexys4 DDR board**.** When write is enabled from the MIPS processor core the display module can be selected when the matching display address is on the bus line, thereby writing the HWDATA bit line (determined by software writes) to the display output peripheral registers (enable, digit, and dp) for decoding on each rising clock edge. The pre-written display timing and decoding modules then take the data registers and decode for the correct display pin outputs that are AN (enables), and segment enables respectively. Only one display can be output at a time, so the timing module continuously loops through each segment display.

AHB-Lite Selection:

The display module is only selected when the bus line address has the value h'7d on bits 28 through 22. If a write is also enabled and the AHB-Lite bus is not idle then once this address is on the line, the current AHB-Lite HWDATA will be written to the display registers and be decoded.

Reset:

The seven-segment display enables, decimal points, and digit registers are all active low decoded, and so on a reset all will be set to high on all bits, thereby initializing all displays to OFF.

Connecting Everything:

Along with writing the display peripheral in Verilog, instantiating a debouncer at the top level of the hardware design was also required, as well as connecting all pins and IO wires to the correct ports so that the display hardware works correctly. The display module outputs the display pin outputs and so had to be wired correctly all the way to the top-level module. The reference hardware project files include both the new display module as well as all modules that had to be modified for project purposes. All new and modified Verilog files can be found for reference under Source/Hardware.

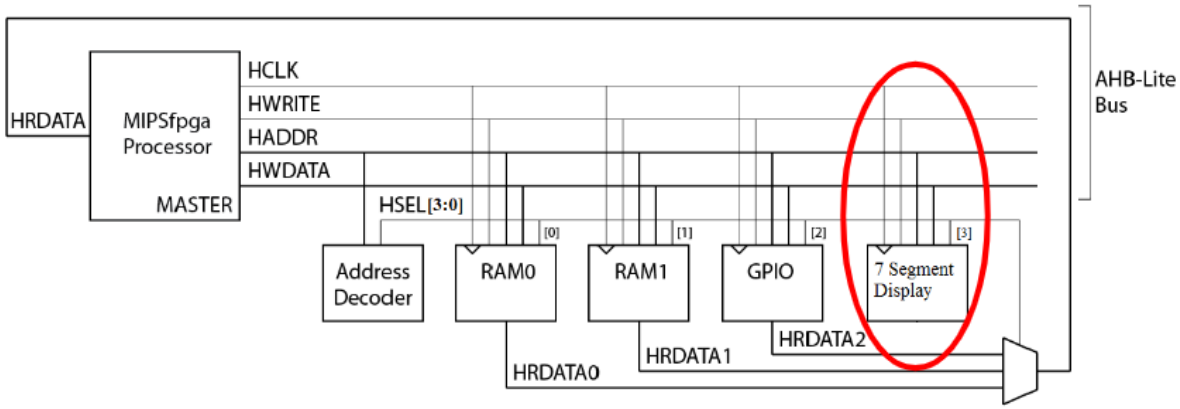

Fig.1: An AHB-Lite(bus) peripheral for 7-segment display of Nexys4 DDR.

II.2 Software Development

Brief Summary of Design:

In software part of this project, a provided MIPS Assembly program has been modified to read input from a combination of buttons on the Nexys4 DDR board and output a campus direction (3 digits display) based on relative turning position of the robot and robot motion(4th digit display) based on button input, per the project requirement shown below.

Push Button	**Motor Function**
BTN UP	Left motor reverse
BTN DWN	Right motor reverse

BTN LEFT	Left motor forward
BTN RIGHT	Right motor forward

Left Motor	**Right Motor**	**Robot Motion Mode**	**Compass Direction**
Stop	Stop	Stop	Idle(g)
Forward(BTN LEFT)	Stop	Turn Right 1X speed	Clockwise
Reverse(BTN UP)	Stop	Turn Left 1X speed	Counter clockwise
Stop	Forward(BTN RIGHT)	Turn Left 1X speed	Counter clockwise
Stop	Reverse(BTN DWN)	Turn Right 1X speed	Clockwise
Forward(BTN LEFT)	Reverse(BTN DWN)	Turn Right 2X speed	Clockwise
Forward(BTN LEFT)	Forward(BTN RIGHT)	Forward	Idle(a)
Reverse(BTN UP)	Forward(BTN RIGHT)	Turn Left 2X speed	Counter clockwise
Reverse(BTN UP)	Reverse(BTN DOWN)	Reverse	Idle(d)

To implement the above system requirements, a finite state machine shown below in Fig.2, has been designed in the provided MIPS assembly program, main.S.

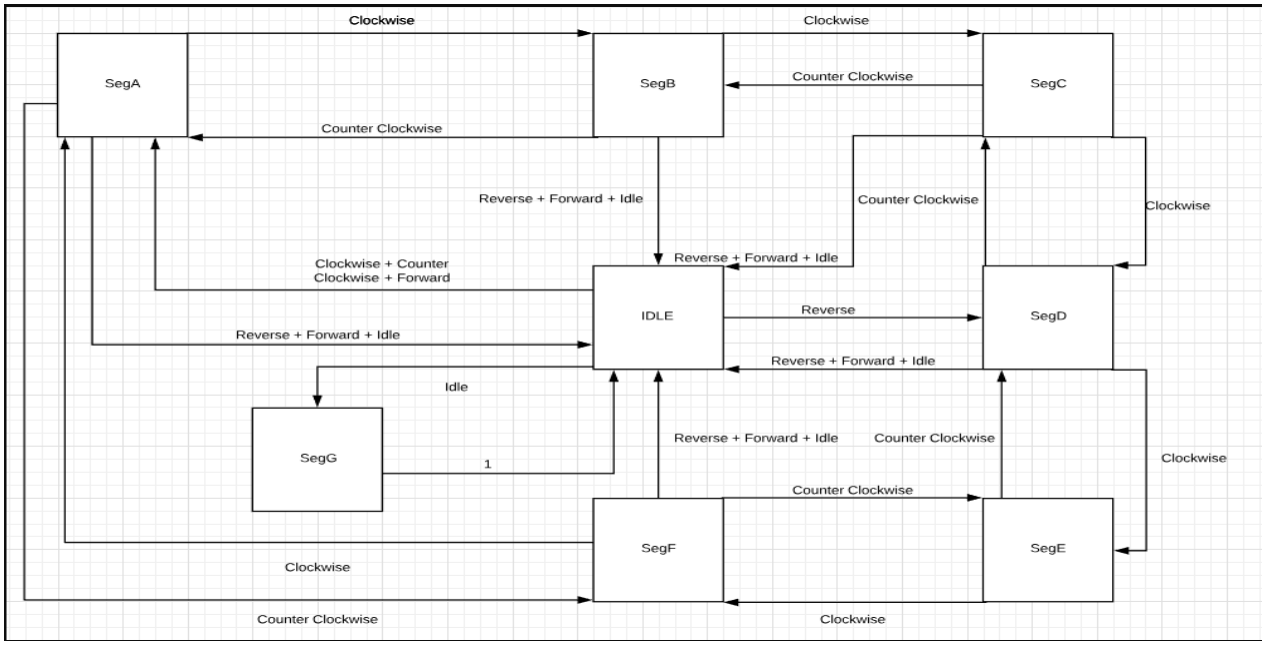

Fig.2: Motion State transition diagram for the SimpleBot project (Moore machine, next state computed entirely from button input).

Because the robot modes have differing frequencies, the software only calculates the next state if the current clock period has been reached. If the clock period has not been reached, the next state logic is skipped and the program returns to the beginning of the ReadIO loop for another check. The clock period to reach depends on the button inputs. Idle, Forward, and Reverse is slow (1hz), while turning can be either 5hz or 10hz depending on the buttons pressed. For the purposes of this project, a loop counter was used to calculate whether or not the correct clock period has been reached. The number of loops for each speed is approximated as described below:

1hz Period = 1 Million ReadIO loops
5hz Period = 200,000 ReadIO loops
10hz Period = 100,000 ReadIO loops

If the clock period has been reached and the buttons indicate that the robot is turning, then the compass needs to be either incremented or decremented. A transition diagram for the compass is shown in fig. 3. Whether or not the compass needs to be updated, if the clock counter has reached the required period, the program always continues onto the next state logic.

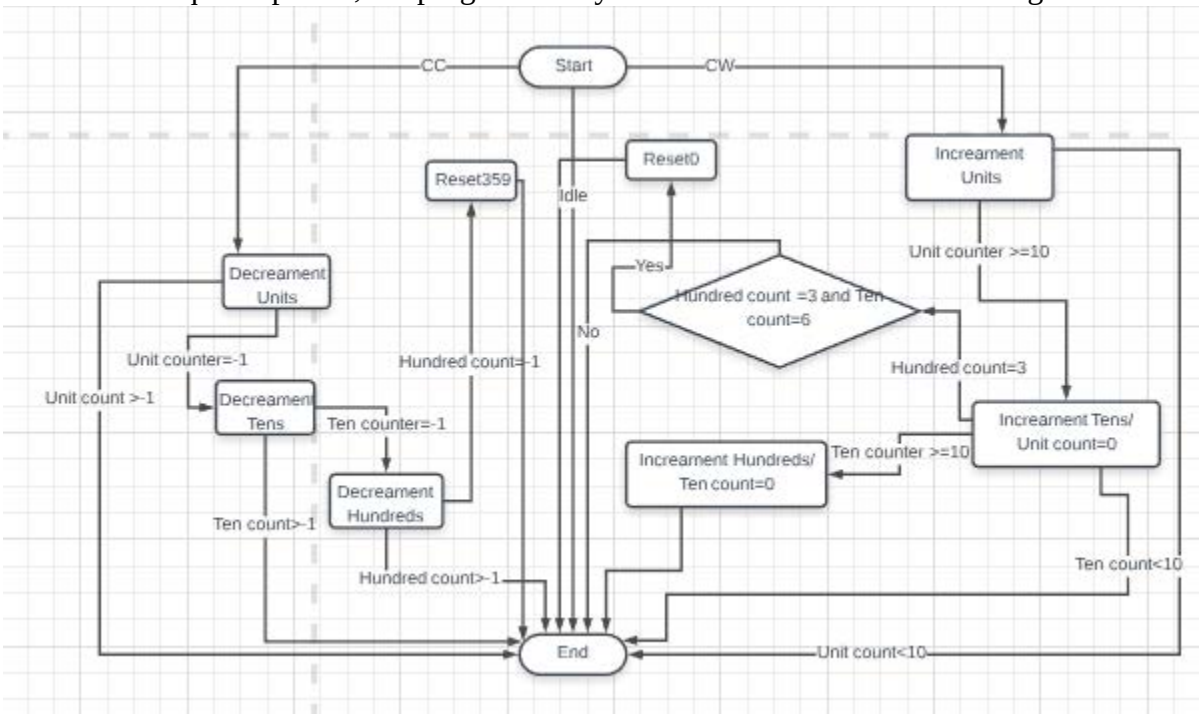

Fig. 3: Transition Diagram for the compass display, CC=Counterclockwise and CW=clockwise.

At the beginning of every ReadIO loop, before updating compass values or calculating the next motion state, the current display data is written out to the display digit address. The current motion data digit value is determined from an output state machine shown in fig. 4. Compass digits are written out based on the current compass registers.

State	Motion Digit Display Segment
IDLE	Blank
A	A
B	B
C	C
D	D
E	E
F	F
G	G

Fig. 4: Motion State Machine Segment Output Table

Detailed Code Design Analysis:
For the following line numbers refer to the main.S file located under Source/Software in project files.

Lines: 43 - 63
The main section of the software initializes all the required address registers and initializes the loop counters as well as initializing the decimal point and segment display registers. The three clock period parameters for 1, 5, and 10 hz are also stored in registers for later checks further in the program. The state machine register is also initialized to zero (IDLE) using register 26.

Lines 72 - 82
This section of the code loads the compass data (stored in three registers for ones, tens, and hundreds) into the digit data register ($3). Each digit is shifted to the correct location in the data. Every digit is spaced 1 byte apart.

Lines 86 - 94
These branch check statements determine which state output code section to jump to in order to add the correct motion output to the digit data register.

Lines 96 - 136
This section of the code is the motion output state logic. Refer to figure 4 for the output by state logic.

Lines 138 - 142
The motion data is added to the digit data register ($3) and is then written to the AHB-Lite Bus to be displayed via the hardware.

Lines 157 - 176

These branch statements determine which clock period to check for determined by button input. If no check is required then the ReadIO loop resets.

10hz Clock Period Check: Robot Turning Quickly
5hz Clock Period Check: Robot Turning Slowly
1hz Clock Period Check: Robot Idling / Moving Forwards / Moving in Reverse

Lines 178 - 213
These code labels check for the different clock periods and if the robot is turning and a clock period has been reached, then the compass is updated.

Lines 215 - 266
Compass Update Logic -> Refer to figure 3.

Lines 268 - 290
If this section of code is reached, then the clock loop counter is reset and branch statements are used to enter the next state logic for the current state (located in register 26).

Lines 293 - 415
Refer to figure 2 for the motion state machine next state logic. The code will then jump to lines 416.. to set the next state register accordingly for the next loop.

Lines 416 - 441
Based on the next state, the state register is set accordingly.

Lines 443 to 448
Increments the loop counter and resets the loop moving back to ReadIO whereby restarting the entire process again.