Main Page

Project Description:
The scenario of the project is that we have a rectangular arena(2.2m*1.35m). The surface of the arena is devided into a grid of leds. Leds are distributed in a 16*16 grid with a resolution of 14cm(length) on the longer axis and 8cm on the smaller axis(width). The leds are considered as nuggets(only those leds which are swiched on). The robot has to collect the nuggets by visiting them. Moreover, the arena also has walls around it. The robot has to have a capability to avoid continues bumping into the wall.
System Architecture:
The idea is to use subsumption architecture to design our project problem based on the behaviors that are required. The main required functionalities are:

- Roam to find nugggets
- goto nugget
- park on nugget
- erase nugget in case robot visits it
- avoid walls
- search nugget
- go home
- untangle wires
- get pose from pose server.

To achieve required functionalities, the following behaviors can be implemented in subsumption architecture.

- Explore-search territory
- Avoid obstacle
- drive to nugget
- home and untangle
- fail-safe-avoid damage

The following components have been developed to achieve the desired behaviors:

- Explorer: To provide a best possible goal to explore nuggets
- Perception: Look for nuggets while the exploration goes on.
- Nugget manager: Keeps track of the nuggets
- Navigator: To provide reactive navigation behavior to robot.

Navigator component:
The goal is to provide navigation, guidance and control for RP6 robot. We have listed the following milestones.

1. Integration of hardware
2. Measurement of the encoder ticks and direction of rotation.
3. Implementation of communication(I2C) between FPGA and Motor control board.
4. Improve measurment encoder to measure velocity as velocity feedback.
5. Implementation of speed controller to command wheel angular velocities and desired bias to move the robot on curved path.
6. Implementaton of User interface to set speed and adjust controller gains.(In progress)
7. Implementation of motion controller to rotate and move to a certain goal.
8. Implementation of I2C communication between FPGA and beagleboard.
9. Implementation and verification of I2C communcation by integrating FPGA, beagleboard and motor control board.
10. Integration of precision controller with speed controller and light sensors.

Milestones:

1. Integration of hardware
    - RP6 Robot
    - FPGA board
    - Motor shield V2 board(motor controller)
2. Measurement of the encoder ticks and direction of rotation.
    - Software implementation in VHDL on FPGA.

- UP/Down counts the ticks of the encoder.
- **Success creteria:** Displays status(on/off) of channel A and B of the encoders on the LED's.
- **Success creteria:** Displays counts(hexa-decimal) on 7-segments.
- **Finish date:** 30.11.2014

3. Implementation of communication(I2C) between FPGA and Motor control board.
   - Problem faced: Motor controller was not replying to the I2C commands.
   - Proposed solution:
     - Check H/W e.g. voltage levels using DMM.
     - Use Oscilloscope to varify signals between two devices.(X)
     - Verify FGPA I2C software using VHDL test bench or (X)
   - **Problem found:** Missing power on the motor controller logic circuits.
   - **Success creteria:** Move the robot by sending I2C-command from FPGA to motor controller board.
   - **Finish date:** 11.12.2014
4. Improve measurment encoder to measure velocity as velocity feedback.
   - Computed speed using encoder counts.
   - To compute the speed, we count the ticks at 1 Hz using a prescaled clock.
   - **Finish date:** 05.01.2015
5. Implementation of speed controller to command wheel angular velocities and desired bias to move the robot on curved path.
   - Speed controller as top module which were utilizing PI controller component.
   - The inputs to the speed controller are:
     - Desired velocity(12-bits positive number) for each wheel(ticks per sec) and
     - Desired biased(12-bits positive number) to move on curved path.
   - The outputs are pwm commands(12-bits postive number) for each motor.
   - Writing a simple P-controller. --> Till 18.01.2015
     - Running controller at 1Hz.
     - Set Kp = 1 (6-bit number in vhdl)
     - Response was unstable.
   - Problems found:
     - In milestone, we measured the encoder ticks as unsigned number.
     - Based on the inputs from the channel A & B, we were only counting up and down as a bit array.
     - Problem 1: We were unable to utilize properly, the count value from encoder component in speed controller.
     - Fixed 1: We fixed the problem by converting our counting of ticks to signed number.
     - Problem 2: We were only counting ticks per sec based on only one count cycle.
     - Fixed 2: We took a running sum of four consective counts and calculted mean counts per sec.
     - Problem 3: Slow rate(1 Hz) of the controller and Quadrature encoder counter.
     - Fixed 3: Increased the rate to 16 hz(approx. 62.5 millisec) of both speed controller and Quadrature encoder counter.
     - Problem 4: Kp gain was too high.
     - Fixed 4: We set Kp < 1. In VHDL, it was not possible to have floating number. So we used right shift operations.
       - Computation of error_in for PI controller
         - error_in_R = desired_angular_wheel_speed - actual_speed_R
         - error_in_L = desired_angular_wheel_speed - actual_speed_L
       - pwm_out = pwm_prev + Kp * error_in
       - Instead of applying right shift on Kp, we applied right shift on error_in to avoid Kp going to 0 in VHDL.
   - Then added integral to make both the wheels slave to each other.
     - pwm_out = pwm_prev + Kp * error_in + Ki * Integral
     - Integral = Actual_speed_R - Actual_Speed_L
     - We tuned Ki which was also less then 1.
     - Problems found:
       - Problem 1: Due to the misalignment of the wheel, the robot was turning left on the long run. It was also effected by the desired speed of the wheels.
       - Possible Solution:
         - We added a constant bias to the Integral equation to fix this issue.
           - Integral = Actual_speed_R - Actual_Speed_L + constant_bias
         - But this solution is not good. Suppose the robot is only rotating then it can effect the rotation of the robot.
   - Then we added a bias_velocity term to move the robot on curved paths.
     - Integral = Actual_speed_R - Actual_Speed_L + constant_bias + bias_velocity
   - **Success creteria:** Robot runs straigt and moves on curve paths.**(See attched Vedio in media).**
   - **Finish date:** 26.03.2015
6. Implementaton of User interface to set speed and adjust controller gains.(In progress)
   - Using Push buttons(PB):

        - To reset(PB0) robot, run/stop robot(PB1 and SW0),
        - To set desired speed and desired rotation bias(PB2).
        - To set controller constants Kp and Ki(PB3)
    - Desired velocities and controller constants are taken using switches(SW7-SW1) to set desired
    - Created a top control module to propagate desired inputs to the navigation controller.
    - Added a way to split the pwm signed into pwm and wheel rotation direction.
7. Implementation of motion controller to rotate and move to a certain goal. (TODO)
    - To convert robot speed linear(V) and angular(W) speeds to compute individual wheel speeds(wr==wl) and desired rotation bias(bias_velocity) for speed controller.
    - The algorithm can be:
        - Linear speeds of wheels: Vr = V + (L/2)W and Vl= V - (L/2)W
        - Angular speeds of the wheels: Wr = Vr/r and Wl = Vl/r
        - Desired rotation bias: bias_velocity = Wr -Wl
    - Compute the distances(mm) to be moved by the robot in x and y direction based on the nugget index and robot global pose.
8. Implementation of I2C communication between FPGA and beagleboard.
    - Done by Devrat.
9. Implementation and verification of I2C communcation by integrating FPGA, beagleboard and motor control board.
    - Yet to be integrated.
10. Integration of precision controller with speed controller and light sensors.
    - Precision controller is implemented but not yet integrated.

Previous Milestone:

- Implementation of speed controller to rotate and move to a certain goal.

Current Milestone:

- Implementaton of User interface to set speed and adjust controller gains.

Software Source:

- Project source codes
    - https://github.com/iuriaa/HWSW-Reloaded

- VHDL I2C master source
    - https://www.eewiki.net/display/LOGIC/I2C+Master+%28VHDL%29#I2CMaster%28VHDL%29-CodeDownload

Demonstration 1 of the Progress

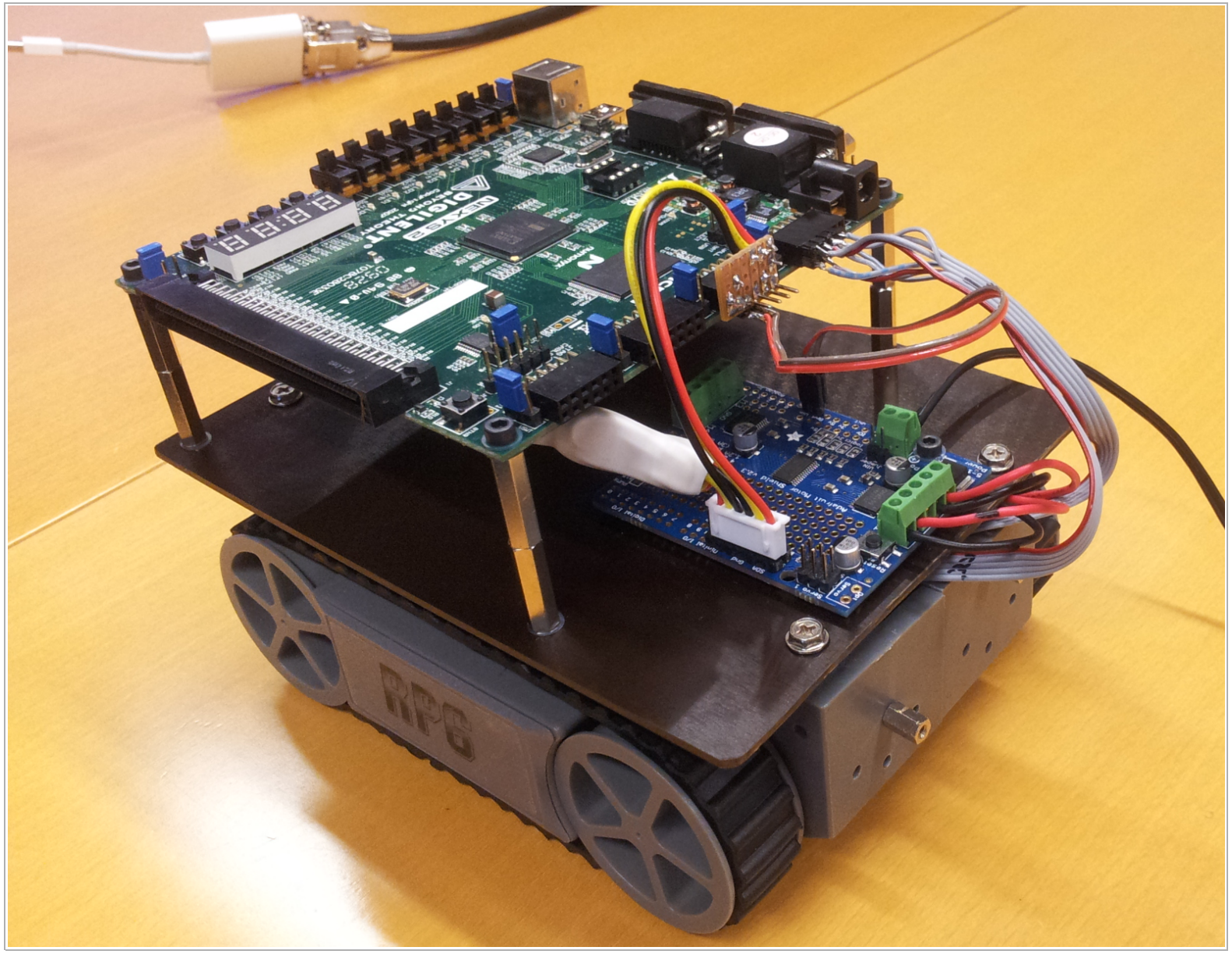
- 16th december 2014

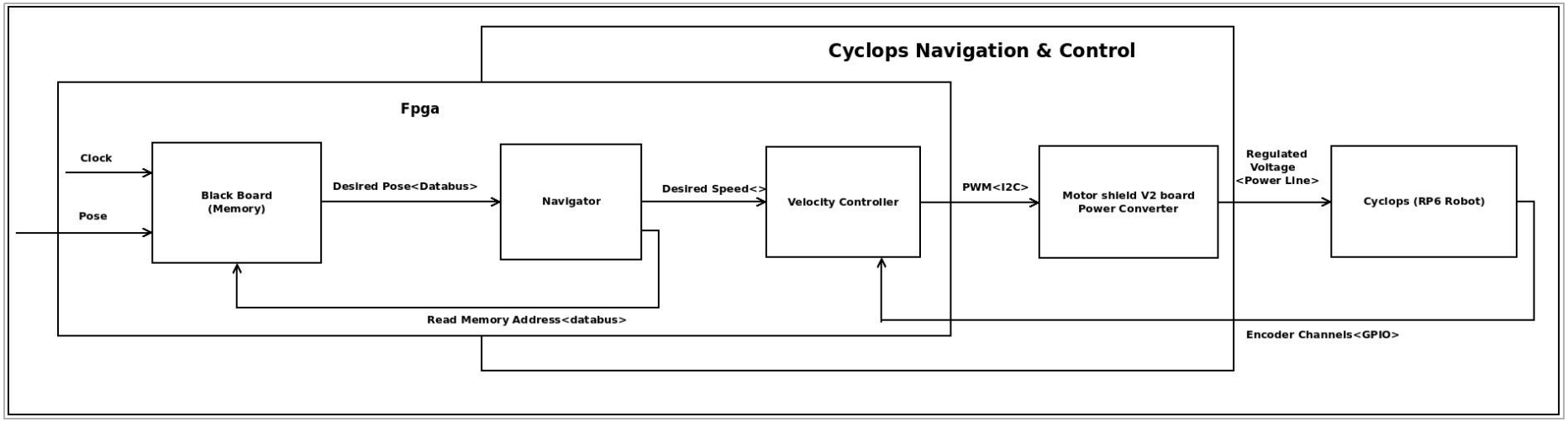
Demonstration 2 of the Progress:

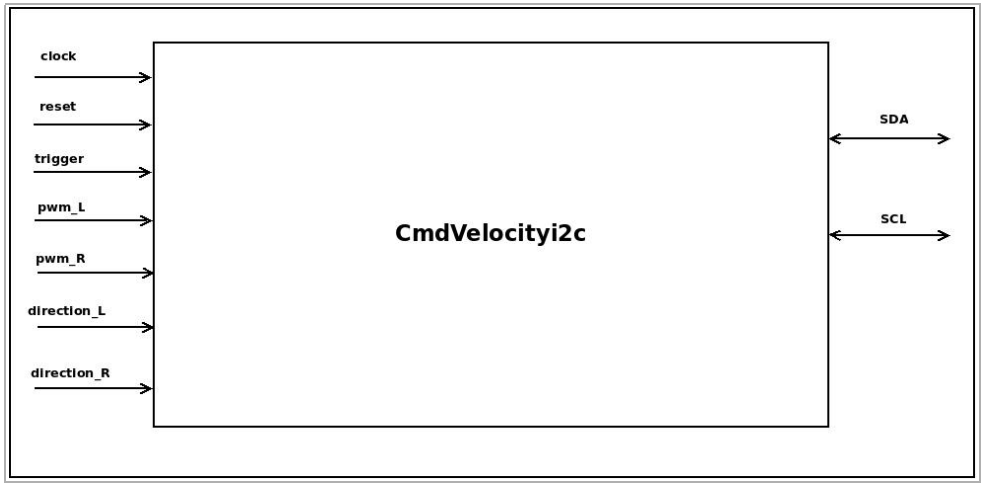
- Last week of the class(18th january 2015)
- ***Suggestions and improvements:***
    - Improve the rate of the speed controller.
    - P-controller was not stable. Problems to investigate: Gain too high and feedback speed(slow rate).
    - Integration of I controller on the feedback path of the controller.

Final examination progress demo.

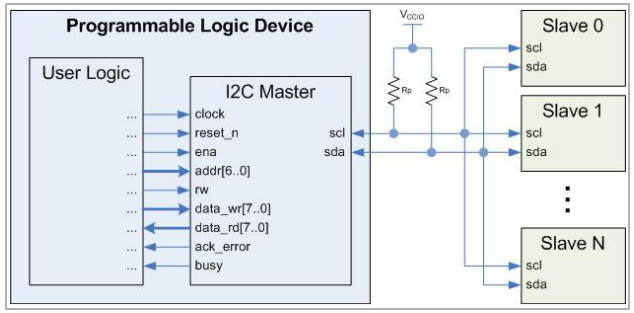
- 26th March 2015


DIGILENT
RPG





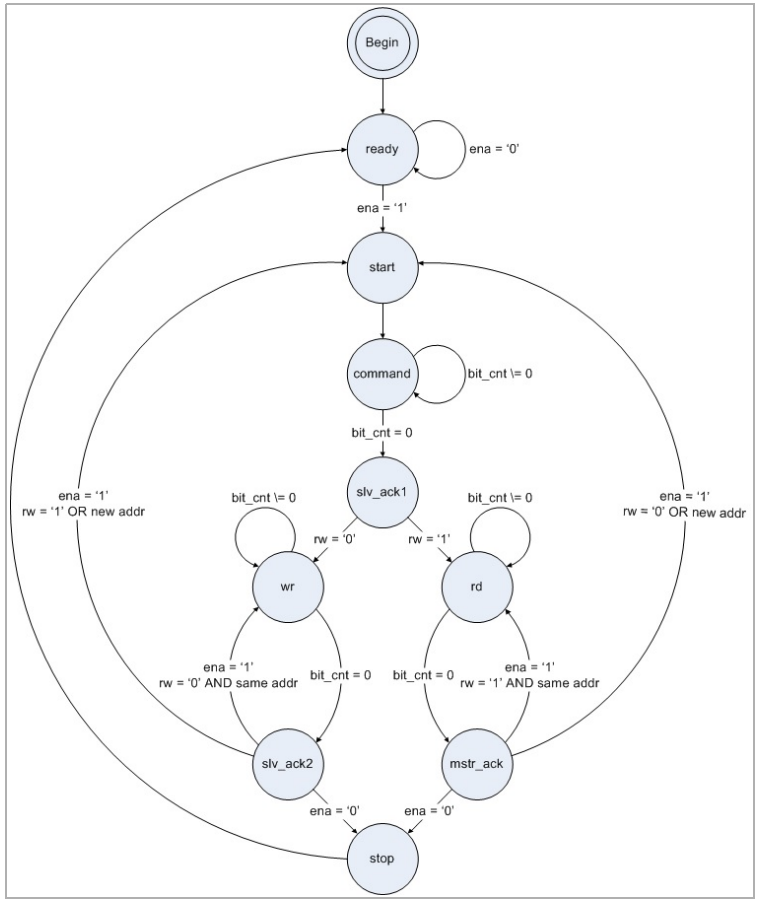
VHDL I2C system diagram : here.

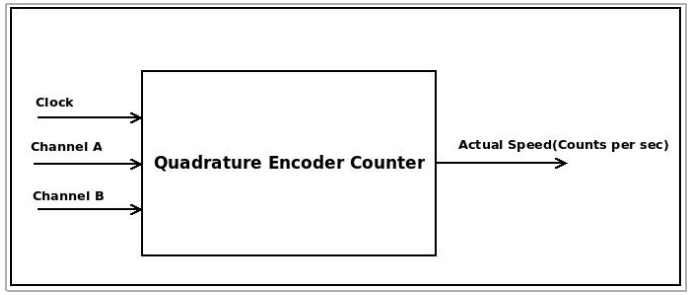


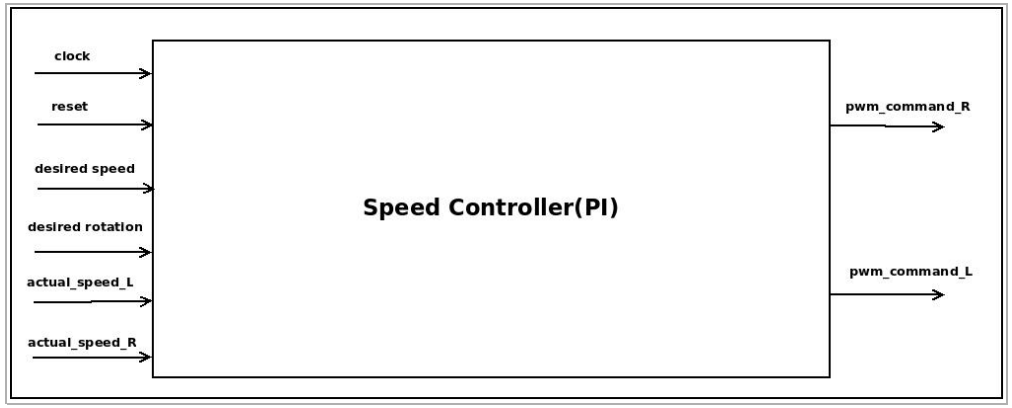
Summary of the state maching used by I2C master:[Reference.]

- Ready state:
  - Component starts at ready state.
  - Waits until the enable signal latches in a command.
- start state:
  - Generates the start condition on the I2C bus
- command state
  - Communicates the address and rw command to the bus.
- slv_ack1 state:
  - Captures and verifies the slave's acknowledgment.
  - Depending on the rw command, the state switches to wr state or rd state.
- slv_ack2 state:
  - On completion, verification of slave's response if writing.
- mstr_ack state
  - Issues its own response if reading.
  - If ena signal Latching a new command:
    - continue another write (wr state) or read (rd state) in case if the command(r/w) did not change.
    - issues a repeated start (start state) in case if the command(r/w) changed.
- stop state
  - Generatation of the stop condition
  - returns to the ready state.

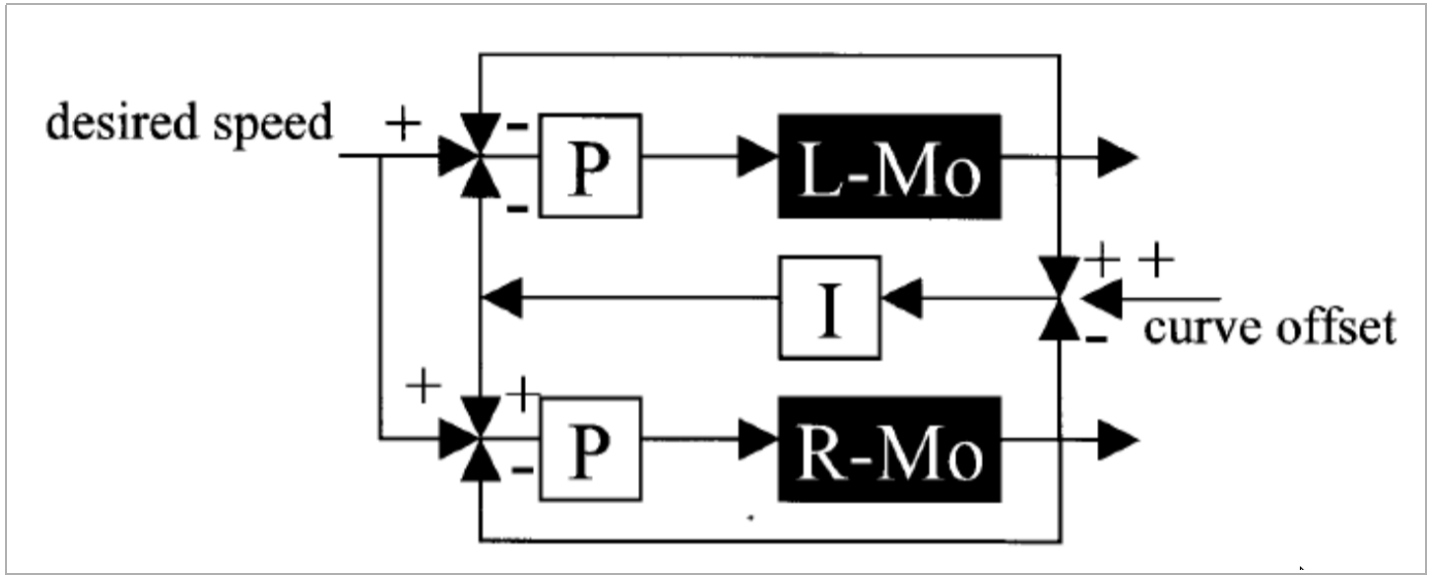
I2C master VHDL state machine diagram : here.


Begin
ready
ena = '0'
ena = '1'
start
command
bit_cnt \= 0
bit_cnt = 0
slv_ack1
ena = '1'
rw = '1' OR new addr
bit_cnt \= 0
rw = '0'
rw = '1'
bit_cnt \= 0
ena = '1'
rw = '0' OR new addr
wr
rd
ena = '1'
rw = '0' AND same addr
bit_cnt = 0
bit_cnt = 0
ena = '1'
rw = '1' AND same addr
slv_ack2
mstr_ack
ena = '0'
ena = '0'
stop


Clock
Channel A
Channel B
Quadrature Encoder Counter
Actual Speed(Counts per sec)



PI controller:[Bräunl, Thomas. *Embedded robotics: mobile robot design and applications with embedded systems*. Springer Science & Business Media, 2008.]



Demo 1: Straight motion with kp

Demo 2: Forward motion with PI

Demo 2: Turn motion with desired bias

00:00
00:00
00:00

Turn_with_bias