ITU Control & Automation Eng. Dept.
KON309E Microcontroller Systems
Final Experiment : Control

Aim: Design a controller for a dynamic system using the HIL technique.

For this experiment, you will need to consult the following reference documents:

- LPC824 user manual
- LPC82x datasheet
- Alakart schematic diagram

Write the code using the peripheral support libraries (Xpressso SDK) or direct register programming or a combination of both. You are welcome to use parts of all previous example code provided for the lectures.

The Experiment

In this experiment, we will be building **a control system on Alakart**. Typically a plant consisting, for example, of a motor with an encoder, motor amplifier and a power supply is provided, and the student is expected to write the embedded software to control it. However, for this class this is not possible because of several reasons:

- We do not have the budget to provide a large number of physical plants
- We do not have control over how you implement the system so we cannot eliminate the risk of your computers being damaged due to miswiring etc.

For these reasons, we will use the **HIL method** which means that instead of a real plant, we will be using a plant model in software. However, the plant model runs on a separate controller (another Alakart) and receives the control inputs $u(t)$ as voltages, and provides output $y(t)$ as voltages. So from the point of view of the controller, the plant appears electrically the same as a physical plant. The detais of HIL method are described in the accompanying document "`HIL.pdf`", so please read that first.

The plant that we will be controlling has a transfer function with a zero at z_1=-2 and two poles at $p_{1,2}=(-1\pm 1)$. It is discretized at 1kHz sampling time and implemented as a firmware that can be programmed into an Alakart board. See the file "`HIL_plant_1kHz/hil_plant.bin`" (a hex file is also supplied). Its source code is also provided in case you want to read it.

The plant has the following properties:

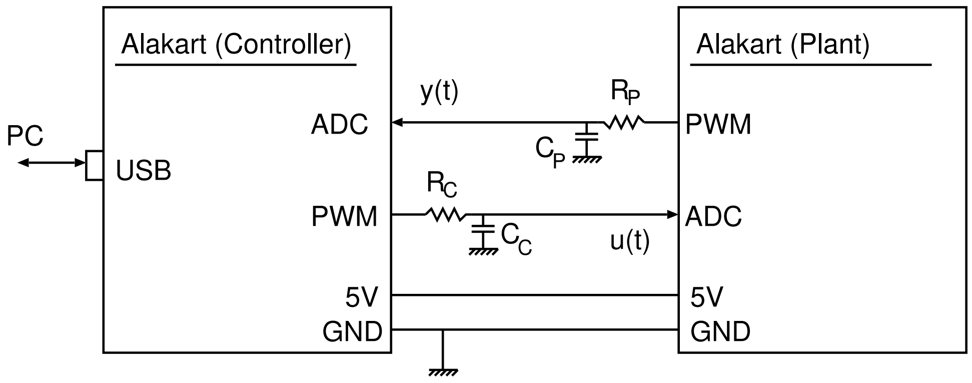
1. Its input is A1 (ADC0_CH1) on PIO0_6, with a voltage range of (0, 3.3)V as usual.
2. Its output is produced as a PWM signal since there is no digital to analog converter on the LPC824. The PWM is produced on pin PIO0_27. PIO0_27 is also the green LED. Its brightness will change with the PWM duty ratio and give you an indication of the plant output magnitude.
3. The PWM frequency is 100kHz, and the bandwidth of the plant is only 1kHz.

Since a PWM signal is used for the plant output, you will need to convert it to an analog signal using a low pass filter. This can easily be implemented as a first order RC circuit. See the circuit diagram below and the file "`HIL.pdf`"

The controller will have similar properties; its input is an ADC input pin (you may choose freely) and its output is a PWM signal at 100kHz. (Again you may choose the output pin freely as you like.)

The Circuit

To build the circuit for the system, **you will need two Alakart boards**; one to run as the controller, and the second to run as the plant. The Plant Alakart will be programmed with the provided file "`HIL_plant_1kHz/hil_plant.bin`" (or .hex), and you are expected to write the firmware for the Controller Alakart. You can see in the circuit diagram that there are two low pass filters to convert the PWM output signals of the plant and the controller to analog voltages.



Only the Controller Alakart is connected to the PC over USB. The Plant Alakart must receive its power from the Controller Alakart. Connect the GND and 5V supply voltage from Controller Alakart to the Plant Alakart as shown in the circuit diagram. Make sure you do not short the power lines as **THEY CONNECT DIRECTLY TO THE POWER SUPPLY OF YOUR PC!**

if you make a mistake here, you may permanently damage your PC. No need to panic; just know what you are doing and it will be fine.

PART I

Design and implement the circuit, configure the ADC and PWM.

1. Calculate the resistor and capacitor values for the RC filter. It must:
    1. Have a cutoff frequency of 3kHz
    2. The values must be chosen out of the E12 values.
       What are E12 values?
2. Obtain the components from your assistant.
3. Construct the circuit diagram shown above. You will need to borrow, beg or steal another Alakart from a fellow group. You may cooperate with another group to share Alakarts. (No we do not have extra Alakarts to give out to all groups.)
   It may be a good idea to build the circuit together and then decide on time slots to take turns for testing.
4. Program the Plant Alakart with the supplied file.
5. Make sure that the **power connection is wired correctly**.
6. Connect the Controller Alakart to your PC.
7. Temporarily disconnect the ADC input of the Controller Alakart (labeled $y(t)$ in the circuit). Connect your potentiometer so that you can supply a desired voltage. See the circuit diagram in the ADC experiment.

You are now ready to program your controller. Write a program that will:

1. **Read an analog port at 100Hz.** You are expected to use the Systick and trigger the ADC conversion by software.
2. Print the result of the ADC conversion to your PC terminal.
3. Produce a simple PWM signal from SCT0 **with a frequency of 100kHz** and a maximum duty ratio of 300 counts.
4. Using SWM, connect the PWM output to a suitable Pin, preferably PIO0_27 (the green LED).
5. At every ADC conversion, update the PWM duty ratio.
6. Repeat

(For this step you do not need the Plant Alakart).

At the end of this step, you will have a PWM signal with a duty ratio that will change as you turn the potentiometer, and the ADC value will be displayed on the terminal. The sampling time of your controller has now been set to **100Hz.**

Store the code for this part in a folder named "EXF_PART1".

PART II

Convert the ADC value obtained in Part I to a floating point value. Write a program that will do the following:

1. Read the ADC value as in Part I
2. Convert the ADC reading with a range of (0,4095) and a variable type of 16 `bit unsigned int (uint16_t),` to a variable of type `float`. Call this variable "`yt`". How to do this conversion is explained in great detail in the document "`HIL.pdf`".
3. Print out yt to the terminal. However, since **it is not easy to print out a floating point number** (the conversion from computer code to human readable is not trivial), we will do a simplification. We only print out to 2 digits after the decimal. To do this:
    1. Create a new `uint16_t` variable.
    2. Make it equal to yt * 100 typecast into `int16_t`.
       (**Do you know how to typecast?**
       It is important for embedded computing!)
    3. Printout the result as an integer. The value will obviously be 100 times the original, but we will "imagine" a decimal point there.
4. This time drive the PWM not from the ADC reading, but from `yt`, which is a floating point number. Again a series of typecasting steps is required. How to do it is described in detail in the file "`HIL.pdf`".

At the end of this step, you will have a PWM signal with a duty ratio that will change **smoothly** as you turn the potentiometer, and the ADC value will be displayed on the terminal as a series of integers **smoothly** changing in the range (-200,200). There is an important difference: The ADC value is now represented as a floating point number within the program, the midpoint of the potentiometer will show '0', and the PWM duty ratio is also driven by a floating point number. If you measure the output of the filter, you should see an analog voltage between (0,3.3)V as the potentiometer is rotated.

Part II positions us where we can code a dynamic PID controller which has a floating point input with positive and negative values, and a floating point output where we again have positive and negative outputs.

Store the code for this part in a folder named "EXF_PART2".

PART III

Finally we are ready to write the controller. You may be familiar with a PID controller in continuous time. However, time for Alakart is not continuous; it advances in short increments. We should convert the continuous time PID equations to discrete time. How to do this is explained in the accompanying document "`PID_digital.pdf`". We should think of the plant output y(t), reference r(t), error signal e(t) and control signal u(t) as discrete time values.

Use the existing code from Part II and the circuit you have built.

Write a program that will do the following:

1. Generate a reference signal that will take the value of '1' and '0' (square wave) with a period of 4 seconds. (2 seconds of '1' followed by 2 seconds of '0')
2. Implement the PID controller in discrete time, using the generated reference and measured ADC values from HIL plant output as inputs. Take the following values as starting point: K_p=1, K_d=0.05, K_i=1.
3. Apply the PID controller output to the plant using the PWM and the filter.
4. Print out the important variables to the terminal.

Take care about signs of the calculations etc.; an error in signs may cause the control loop to have positive feedback and of course unstable.

At this step, you should have a HIL control system where the plant output voltage follows the reference. You can track this using a votlmeter or oscilloscope[1].

Finally, adjust the PID values so that the plant output follows the reference signal better. If you implement the Bonus section below and visualize the signals, it will be easier and more pleasant.

Store the code for this part in a folder named "EXF_PART3".

1 Note that the voltage we measure by a voltmeter at the filter output will not be equal to the reference (i.e., you will not see '1V' or '0V'). You will see a transformed voltage; it will be offset by 1.65V (=3.3V/2) and multiplied by the plant output gain conversion (1.65/2=0.825). So if the reference is '0', you should measure 1.65V, and for '1', you should measure about 2.5V (=1*0.825+1.65)

Bonus

Again, this is the fun part. Display the reference and plant output on the screen graphically. There are several software tools for this. We recommend using SerialPlot by H. Y. Ozderya. It is an easy to use and highly configurable program. A demonstration is already given in the class.

You do not need to provide a code for this part; it will be evaluated by the class assistant.

A bonus of +5 points will be given.

Write in your report:

- What values must be written to which registers so that the I^{2}C peripheral is configured correctly?
- How did you calculate the low pass filter values?
- The frequencies of "main clock" and "system clock" by checking the configuration functions. Write which functions you check and how you find the result.
- How is the ADC configured to run at **100Hz** and the **PWM at 100kHz**?
- Any other item of significance, such as the structure of the code and the problems that you have had during the development.

Report Content

Your report must be a formal account of what you did in the experiment. Make sure that it includes the following items:

- How you have configured the processor i.e.:
  - What pins are inputs, what pins are outputs, etc.
  - Which peripheral devices were explicitly powered up,
  - How the GPIO and other peripherals were configured,
  - What values were written to which registers,
  - Which default settings of the peripherals were used.

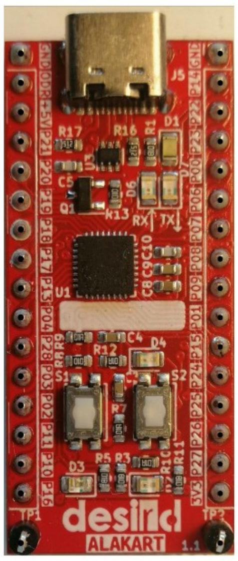
These must appear in all of your reports.

- **Any inclusion of code must be as formatted text.**

(Photographs or screenshots are not accepted!)

Appendix1: Pin connections of components on Alakart

- LEDs:
  - RED: D4 on GPIO PIN12
  - Blue: D3 on GPIO PIN16
  - Green: D2 on GPIO PIN27
  - White: D1 Power on
  - Green: D6 Transmit to PC
  - Red: D7 Transmit from PC
- Buttons:
  - S1: Reset
  - S2: ISP (enter boot mode) Also: User button.
- Red Pins: Test
  - TP1: GPIO PIN16
  - TP2: GPIO PIN27
- Purple Pin: Open drain FET
  - ODR: GPIO PIN21

GND		GND
ODR		P14
+5V		P22
P21		P23
P20		P00
P19		P06
P18		P07
P17		P08
P13		P09
P04		P01
P28		P15
P03		P24
P02		P25
P11		P26
P10		P27
P16		+3V3
TP1		TP2

Note:

- PIO0_2, and PIO0_3 are debugger pins by default. PinPIO0_5 is Reset pin by default. You can use them if you wish, but will **need to disable their default functionality in PINENABLE0 register** first.