Chapter 1
Course
Introduction/Overview

©2017 Mark Wickert

Contents

1.1 Lecture Outline ........................................ 1
1.2 This Course and the Phase-Locked Loop Landscape .... 2
  1.2.1 General PLL Perspective .......................... 2
  1.2.2 Course Topics ..................................... 3
1.3 Course Perspective in the Comm/DSP Area of ECE .... 5
1.4 Computer Analysis/Simulation Tools ...................... 6
1.5 Instructor Policies ....................................... 7
1.6 Course Syllabus .......................................... 8
1.7 Required Student Background ........................... 9
1.8 References .............................................. 10
  1.8.1 General PLL ........................................ 10
  1.8.2 Synchronization .................................... 11
  1.8.3 Journals ............................................ 11
1.9 PLL Highlevel Model .................................... 12
1.10 Four Case Studies ...................................... 15
1.1 Lecture Outline

- This Course and the phase-locked loop (PLL) Landscape
  - General PLL perspective
  - Course Topics
- Course perspective in the comm/DSP area of ECE
- The role of computer analysis/simulation tools
- Instructor policies
- Course syllabus
- Required student background
- References
  - Books
  - Reports
  - Journals
- PLL introduction and synchronization applications overview
1.2 This Course and the Phase-Locked Loop Landscape

1.2.1 General PLL Perspective

- The focus of this course is phase-lock loops (PLLs) and synchronization applications

- At first this may seem like a very narrow course of study, but the PLL has many applications and many implementation variations

- The use of PLLs for frequency synthesis, i.e., creating a stable yet tunable local oscillator for radio transmitters and receivers is one traditional application area

- In communication systems in general the PLL is widely used for synchronization:
  - Carrier phase and frequency tracking
  - Symbol (bit) synchronization
  - Chip synchronization is spread-spectrum systems (this includes GPS receivers)

- Clock recovery is another related topic (same class of problems as symbol sync)

- The implementation may be:
  - All analog electronics (microwave/RF/baseband)
  - A hybrid of analog and digital electronics
– A hybrid of analog and software
– Pure software

• The implementation technology may be:
  – Board level using RF and baseband devices
  – Single chip with a few off-chip or maybe no off-chip parts
  – Custom ASIC or FPGA
  – A combination of RF and baseband analog with the remainder in software via a real-time digital signal processing
  – Entirely real-time DSP approach if signal samples are acquired say using an asynchronous sampling clock; this is the current state of technology

1.2.2 Course Topics

• PLL fundamentals
  – Loop components
  – Loop response
  – Loop stability
  – Transient response
  – Modulation response

• Discrete-Time PLL’s

• Performance in noise
  – Input noise
Chapter 1 Introduction and Overview

– Phase noise
– Nonlinear behavior and cycle slipping

• Acquisition
  – Unaided
  – Aided

• Analog PLL lab experiment (is there interest?)

• Communication applications
  – DSP-based carrier phase tracking algorithms
  – DSP-based symbol timing tracking algorithms
  – Spread Spectrum chip tracking as found in GPS and elsewhere
1.3 Course Perspective in the Comm/DSP Area of ECE
1.4 Computer Analysis/Simulation Tools

- Pencil and paper will work for many problems
- The use of Python (version 3.6 or 2.7 if you insist) and the scipy stack will be emphasized for:
  - Analysis and plotting
  - Symbolic solutions using sympy
  - Simulation
- The new Python package scikit-dsp-comm will be used, and hopefully by the end of the semester more functions will be added to the synchronization module
  - The project can be found on GitHub at https://github.com/mwickert/scikit-dsp-comm
  - Installation instructions and tutorial material from Scipy2017 avn be found at https://github.com/mwickert/SP-Comm-Tutorial-using-scikit-dsp-comm
  - Documententation is being build on readthedocs at http://scikit-dsp-comm.readthedocs.io/en/latest/?badge=latest
  - A set of example Jupyter notebooks is under construction at https://mwickert.github.io/scikit-dsp-comm/
1.5 Instructor Policies

- Working homework problems will be a very important aspect of this course
- Each student is to his/her own work and be diligent in keeping up with problem assignments
- If work travel keeps you from attending class on some evening, please inform me ahead of time so I can plan accordingly, and you can make arrangements for turning in papers
- The course web site: http://www.eas.uccs.edu/~mwickett/ece5675/
  will serve as an information source in between weekly class meetings
- Please check the web site updated course notes, assignments, hints pages, and other important course news
1.6 Course Syllabus

ECE 5675/4675 Phase-Locked Loops and Digital Communication Synchronization

Fall Semester 2017

Instructor: Dr. Mark Wickert
Office: EN-292
Phone: 255-3500
Fax: 255-3589
mwickert@uccs.edu
http://www.eas.uccs.edu/~mwickert/ece5675

Office Hrs: Monday 8:00 – 10:40 am and after class as needed, others by appointment.
Note: These hours may be adjusted if needed.

Required Texts:

Optional Texts:

Required Software:

Grading:
1.) Graded homework assignments including computer tools totaling 40%.
2.) Mid-term Exam worth 25%.
3.) Analog and/or DSP-based PLL Laboratory 10%.
4.) Final Project/Exam worth 25%.

<table>
<thead>
<tr>
<th>Topics</th>
<th>Text Chapters and Notes</th>
<th>Session (weeks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction/Overview</td>
<td>1 &amp; notes</td>
<td>1.5</td>
</tr>
<tr>
<td>2. Initial acquisition and frame synchronization</td>
<td>2 &amp; 3 &amp; notes</td>
<td>1.5</td>
</tr>
<tr>
<td>3. Phase-locked loop fundamentals; analog &amp; digital</td>
<td>4 &amp; notes</td>
<td>3.0</td>
</tr>
<tr>
<td>4. Analog PLL lab experiment*</td>
<td>Handout</td>
<td>1.5</td>
</tr>
<tr>
<td>5. PLL tracking performance in noise &amp; phase noise</td>
<td>4 &amp; notes</td>
<td>2.0</td>
</tr>
<tr>
<td>6. Unaided and aided acquisition</td>
<td>Notes</td>
<td>1.0</td>
</tr>
<tr>
<td>7. Carrier synchronization</td>
<td>5 &amp; notes</td>
<td>1.0</td>
</tr>
<tr>
<td>8. Timing synchronization</td>
<td>6 &amp; notes</td>
<td>0.5</td>
</tr>
<tr>
<td>9. Timing control with digital resampling</td>
<td>7 &amp; notes</td>
<td>1.0</td>
</tr>
<tr>
<td>10. RTL-SDR DSP-based PLL/synchronization experiment*</td>
<td>Handout</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Requirements: A background in basic communication theory, probability and random variables, and basic digital signal processing, i.e. sampling theory is desired. Please contact Dr. Wickert if you are considering this course, but are in doubt as to whether you have adequate background. Items with * will be selected by class interest.
1.7 Required Student Background

- Basic linear systems theory is a must
- Random variables is needed for noise analysis
  - A brief introduction to random processes will be provided if needed
- Basic modulation theory is also assumed
- A knowledge of digital communication systems is desirable
- A basic understanding of digital signal processing is required
- Knowledge of sampling theory is needed for digital loop concepts
- Knowledge of $z$-domain concepts is required
- The ability to program using Python, particularly the Jupyter Notebook (http://jupyter.org) is important for all computational aspects of the course; sample notebooks will be posted on the course Web Site
1.8 References

The following list of references is not exhaustive by any means, but is a list of core books I have in my library or have been recommended to me. Some of these books are hard to find since they are now out-of-print.

1.8.1 General PLL


   - This text is basically concerned with analog PLLs (including charge-pump) starting from the very basic concepts all the way through very detailed nonlinear analysis with noise
   - The text also includes material on automatic frequency control (AFC) and automatic gain control (AGC)
   - The book is clearly telecommunications based since PLL synthesizers are not considered at all
   - Used as the course text in earlier (1990’s offering’s of the course)


1.8.2 **Synchronization**


1.8.3 **Journals**

1. IEEE Transactions on Communications.

2. IEEE Journal on Select Areas in Communications.

3. Other IEEE Journals
1.9 PLL Highlevel Model

- In Communication Systems I (ECE 4625/5625) you learn about the basic analog PLL for tracking the phase of a carrier signal:

![Diagram of basic analog PLL (APLL) for tracking the phase of a carrier signal.]

A big assumption for linearity to hold is that the loop is locked with a small tracking error.

- In simple terms the PLL as see above is a nonlinear feedback control system, which can be linearized under suitable assumptions.

- By inserting an analog-to-digital converter (ADC or A/D) the APLL can be moved into the discrete-time domain with a change of components.
Figure 1.2: Basic digital PLL (DPLL) for tracking the phase of a discrete-time carrier signal.

- Fundamentally the PLL is not restricted to carrier phase tracking.

- When the PLL is used to track other waveform attributes, such as sampling instant under the control of an interpolator, or the time of delay of a locally generated waveform, as in spread-spectrum, the PLL concept still holds:

Figure 1.3: High-level PLL block diagram.

- What goes on inside the error detector may be very simple or complex. The requirement of the error detector is provide an $s$-shaped characteristic that under feedback control can be used.
to drive the tracking error to zero (with noise to have mean zero and small variance)

\[ e(t) \text{ or } e[n] \]

S-Curve

Error Between Input Waveform Attribute and Waveform Generator

Slope at zero

= \( K_{ED} \)

so we can linearize

Figure 1.4: The general tracking characteristic or \textit{S-curve}. 
1.10 Four Case Studies

- To close out this introductory chapter we now consider four Python simulation examples

Example 1: Pilot Carrier Phase Tracking in Broadcast FM

- Recall that broadcast FM is a form frequency division multiplexing, with composite baseband spectrum being composed of a collection of useful signals.

![Power Spectrum in dB](image)

Figure 1.5: Received baseband spectrum (following FM demod).

- The receiver block diagram sitting behind a software defined radio (SDR) front-end is:
**Chapter 1 Introduction and Overview**

**Part IV Developing Algorithms for a Broadcast FM Stereo Receiver**

**Problem 1**

Choose an FM station, KCME is fine, and capture 5-10 s from the RTL-SDR. You may use one of your previous captures.

- The clean KCME capture used earlier will be used again as source in for stereo demodulation

\[ x[n] = x_I[n] + jx_Q[n] \]

**Problem 2**

Design a bandpass filter to extract the subcarrier signal from the discriminator output. Plot the frequency response in dB to be confident you have a reasonable design.

- The m-file fm_rx.m contains both mono and stereo demodulation code

---

![Diagram of FM Demod and Stereo Demultiplex](image)

**Figure 1.6: Demodulator for FM stereo recovery.**

- Track the 19kHz pilot phase, \( \theta[n] \), using a PLL (signal was captured live)

- The coherent carrier needed to demodulate the 38 kHz L-R subcarrier is \( \cos(2\theta[n]) \) and the 57 kHz RDS carrier is \( \cos(3\theta[n]) \)

```python
fs = 2400000
b = signal.firwin(64,2*200e3/float(fs))
# Filter and decimate (should be polyphase)
y = signal.lfilter(b,1,x)
z = ss.downsample(y,10)
# Apply complex baseband discriminator
z_bb = sdr.discrim(z)

z_bb19 = signal.lfilter(b19,1,z_bb)
theta, phi_error = sdr.pilot_PLL(z_bb19,19000,fs/10,2,100,0.707);
```
subplot(211)
plot(1000*arange(len(phi_error[:1000]))/(fs/10),phi_error[:1000]*180/pi)
ylabel(r'Phase_$(deg)$')
xlabel(r'Time_$(ms)$')
grid()
subplot(212)
psd(cos(2*theta),2**12,2400/10);
psd(signal.lfilter(b38,1,z_bb19**2),2**12,2400/10);
ylabel(r'PSD$_{(dB)}$')
xlabel(r'Frequency$_{(kHz)}$')
ylim([-80,5])
tight_layout()

Listing 1.1: Demodulate and then track the 19 kHz pilot subcarrier.

![Phase transient](image1.png)

Loop transient as it locks to the 19 kHz pilot

![PSD](image2.png)

With and without a bandpass prefilter centered on 19 kHz

Figure 1.7: Traced phase in time and frequency.
Example 2: Bit Synchronization for FSK Demod Signal

- When an asynchronously sampled bit stream is output from an FM discriminator bit synchronization is needed to take samples of the waveform at the point of maximum signal-to-noise ratio

- The sample-correlate-choose-smallest (SCCS) algorithm is used here.

![Diagram of FSK Demodulator Including Bit Synch]

Figure 1.8: FSK demod block diagram with SCCS bit synch.

- Of interest here is optimum sampling index when there are nominally $N = 20$ samples/bit (1 kbps) and $N = 8$ samples per bit (10 kbps)

---

The transmit clock and receiver sampling rate clock are sliding past each other, so over time the absolute sampling index must adjust.

Figure 1.9: Sampling index tracking over time at 1 kbps and 10 kbps.
Example 3: Spreading Code Tracking using a Delay-Locked Loop

- In a spread-spectrum system the received must implement a local spreading code generator that finds coarse code alignment and then tracks the cross-correlation between the received signal code and the local code:

![Diagram of spreading code tracking]

For the GPS CA code $1/T_c = 1.023$ Mcps

$T_s, T_s + \delta t$

Satellite Transmitted CA Code at SV time

User Received CA Code (delayed)

Replica CA Code at local time

Replica CA Code cross correlated at local time

$t_u, t_u - t_{peak}$

pseudo range when scaled by $c = \text{velocity of propagation}$ (includes local clock errors)

As the code repeats cross correlation triangle peak also repeats every 1ms (1023 chips)

$t_{peak} = T_u + t_u$

Figure 1.10: For GPS user time delay measurement using cross correlation with the local replica code.

- For live capture or simulation purposes the following block diagram describes the system with an emphasis on the spreading code tracking:
Chapter 1 Introduction and Overview

ECE 5650/4650 Computer Project #2: DSP in GPS Signal Acquisition and Tracking

Background Theory

The geometric range \( r \) between the satellite and the user is written in terms of time parameters

\[
\text{Geometric Range} = r = c(T_u - T_s) = c \Delta t
\]

where \( T_s \) is the time the signal leaves the satellite, \( T_u \) is the time the signal arrives at the receiver, and \( c \) is the velocity of propagation

- The pseudorange is given by

\[
Pseudorange = \rho = c \left[ (T_u + t_u) - (T_s + \delta t) \right] \\
= c(T_u - T_s) + c(t_u - \delta t) = r + c(t_u - \delta t)
\]

where \( t_u \) is the receiver clock offset or error relative to the system clock and \( \Delta t \) is the offset of the system clock from the true system time

- A delay-locked loop (DLL) is a PLL-like structure for tracking the code and then obtaining the pseudo range

Figure 1.11: Generation of GPS L1 signals similar to live capture from an RTL-SDR.
Chapter 1 Introduction and Overview

ECE 5650/4650 Computer Project #2: DSP in GPS Signal Acquisition and Tracking

Background Theory

A portion of a carrier tracking system, which is often used to remove all of the implemented functionality of Figure 6 is implemented by the Python class CA_code_track found in the module GPS.py (see the appendix of the code ZIP package for more details). The code tracking class also makes use of the code_NCO class. Of special note, each instance of the CA_code_track class is an object containing three code correlators. The prompt or P channel is used to detect coarse code alignment, and in a complete GPS receiver, not implemented here, recover the 50 bps data stream. The early or E and late or L channels are used to form an error signal that advances or retards the code numerically controlled oscillator (NCO). The default time separation between these signals is 0.5 chip. The objective is to keep the replica code perfectly aligned with the input signal. When this happens the NCO code phase contains the pseudorange, which as you know from an earlier discussion, is used to solve for the user location.

All of the signal points marked with a green dot are test points returned by the class attribute trk_var:

```python
#+++++++++++++++++++++++++++++++++++++++++++++++++++++
trk_vars = a collection loop signals recorded during the simulation:
row 0 = abs(early correlator output)
row 1 = abs(prompt correlator output)
```

Figure 1.12: DLL block diagram

- Notice three correlators (local code multiply and 1ms accumulator) at early, prompt, and late time offsets
- The DLL error detector is formed using the early and late correlator outputs
- During the code acquisition phase, when the local code is stepped past the received code, the prompt correlator output is monitored to see if coarse code alignment is found
- By opening the tracking loop we can observe the local code slide past the received code, and generate verify the S-curve
Noting that the code steps 0.05 chips/ms, it is possible to rescale the x-axis to have units of chips.

Moving on, $T_1$ is now set back to 2000 and $\text{ss\_step}$ is set back to -0.05 chips. Now coarse acquisition occurs and the track mode is entered. More plots will be shown as both the tracking

Note: Early is to the right since we are stepping the code backwards.

Figure 1.13: The correlator outputs under open loop tracking.

Figure 1.14: The DLL S-curve, theory and measured.
• Closing the loop we can observe coarse acquisition as the local
is stepped into alignment with the received code

Figure 1.15: With feedback re-enable the three correlator outputs
form the desired E, P, and L outputs.

• When the prompt correlator output crosses a threshold the DLL
is engaged and the loop pulls in and finally begins tracking

• As with any PLL there is some settling time while the loop
error approaches zero
The DLL pulls in and the error settles to zero.

Figure 1.16: DLL pulse-in and track