BWRC

EE290C Final Project: GPS Receiver

By: Daniel Grubb, Olivia Hsu, Zhaokai Liu, Aviral Pandey, Dinesh Parimi, Ruocheng Wang, Zhongkai Wang, Nick Werblun

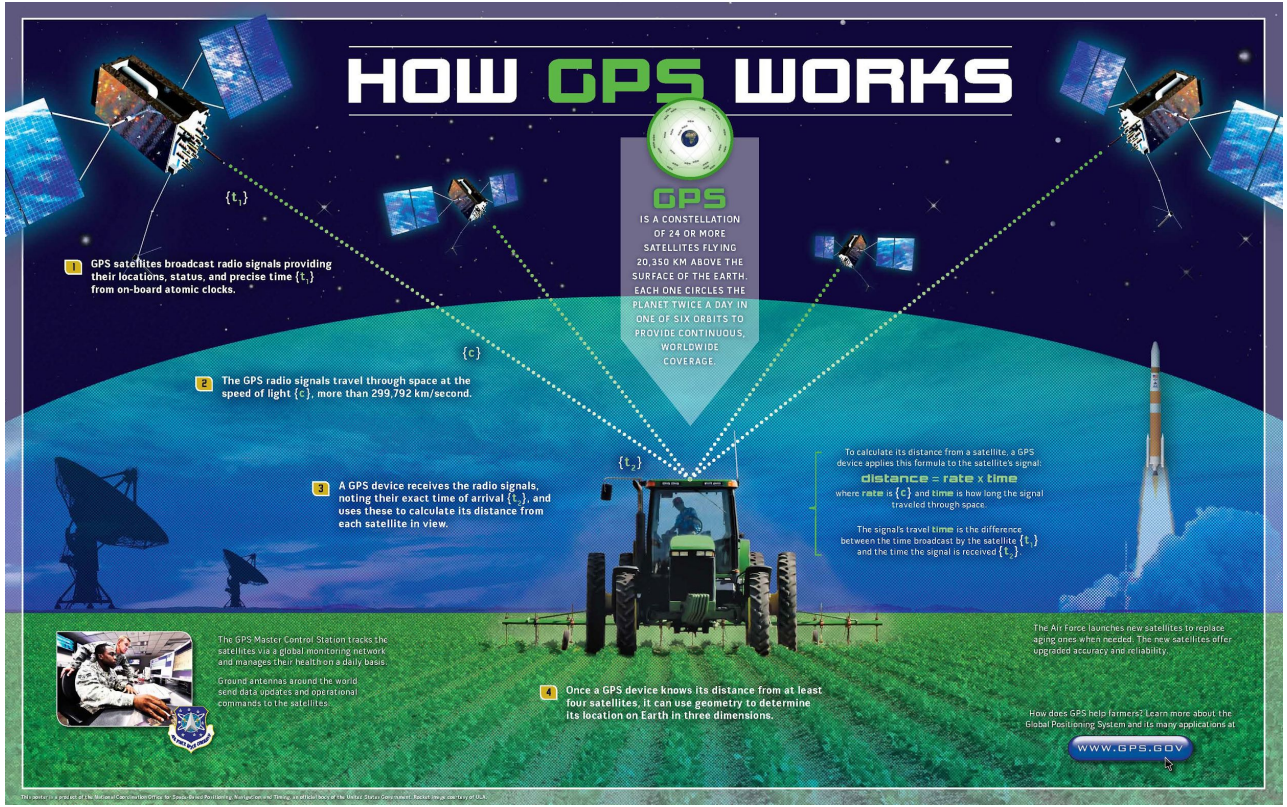
BWRC

Outline

- Overview
  - Modulation
  - Satellites
- Receiver structure
  - Acquisition
  - Tracking
  - Ranging
- Integration
- Conclusion
  - Next steps

Overview

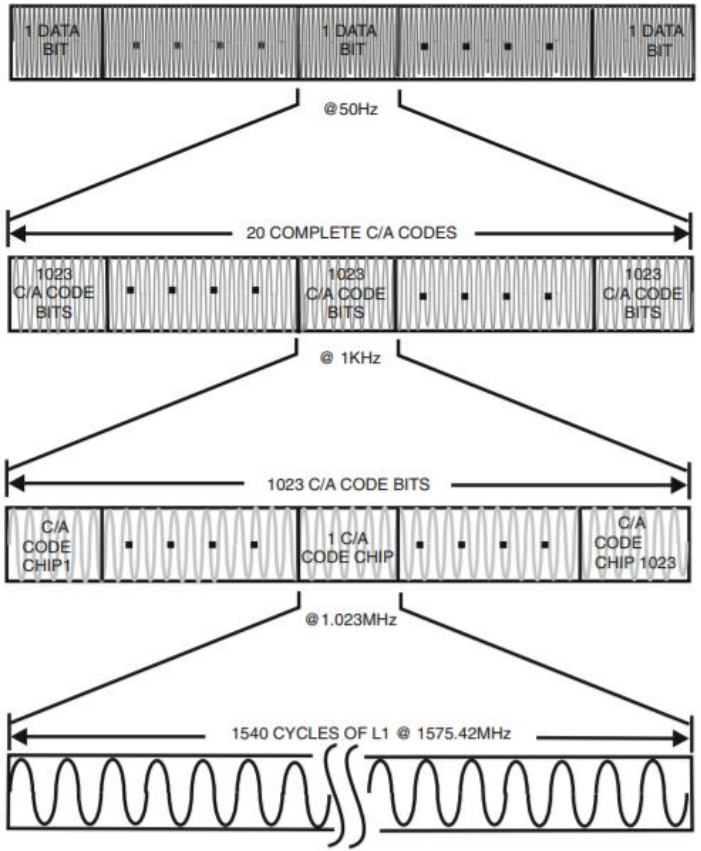
Overview



Source: https://www.af.mil/News/Article-Display/Article/757533/gps-registers-most-accurate-signal-yet/

GPS Signals

- L1 C/A GPS system
- CDMA scheme
- 1023 length PRN codes
- 50 Hz data
- Info both in message and timing

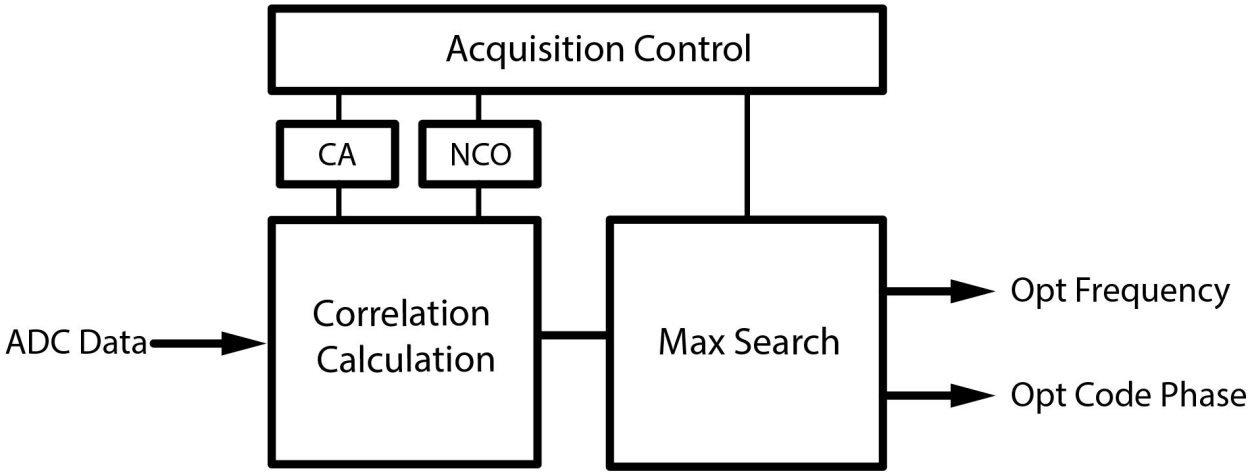


Doberstein. Fundamentals of GPS Receivers, Springer. 2012.

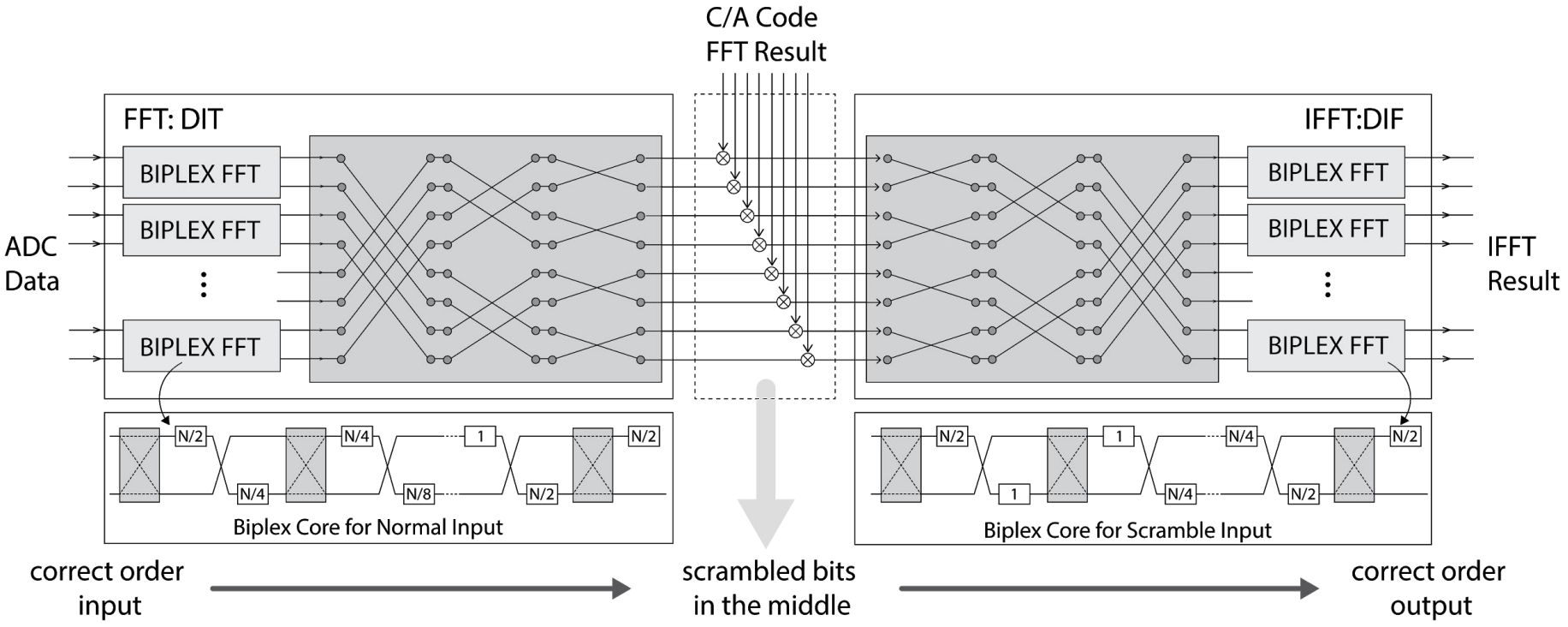
Acquisition

Acquisition: Overview

- Tracking loop has limited frequency and code phase range
  - Need acquisition block to narrow down frequency and code phase
- Get correlation for different frequency and code phase
  - Two methods: FFT and serial correlation calculation



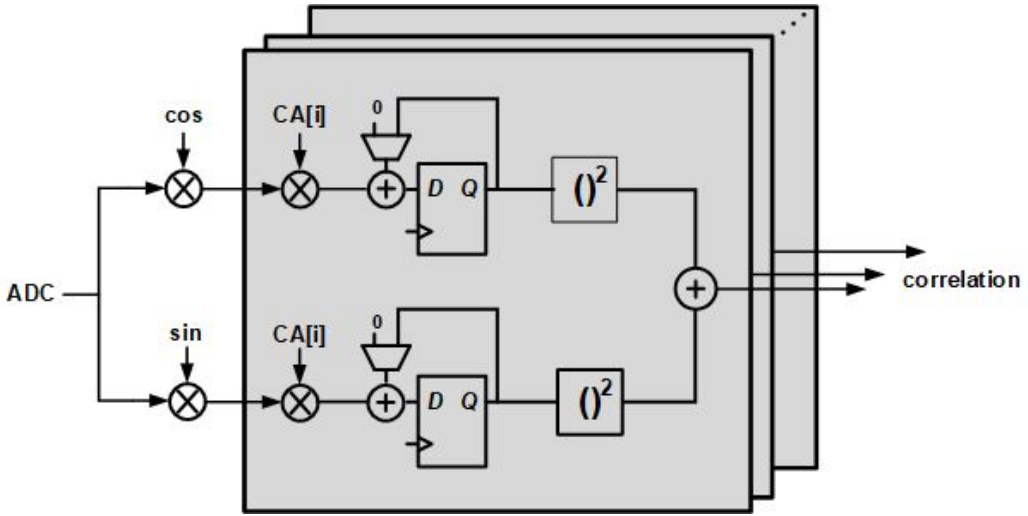
Acquisition: FFT Search



$$R[m] = x[n] \otimes CA[-n] = \mathcal{F}^{-1}(\mathcal{F}(x[n]) \cdot \mathcal{F}(CA[n])^*)$$

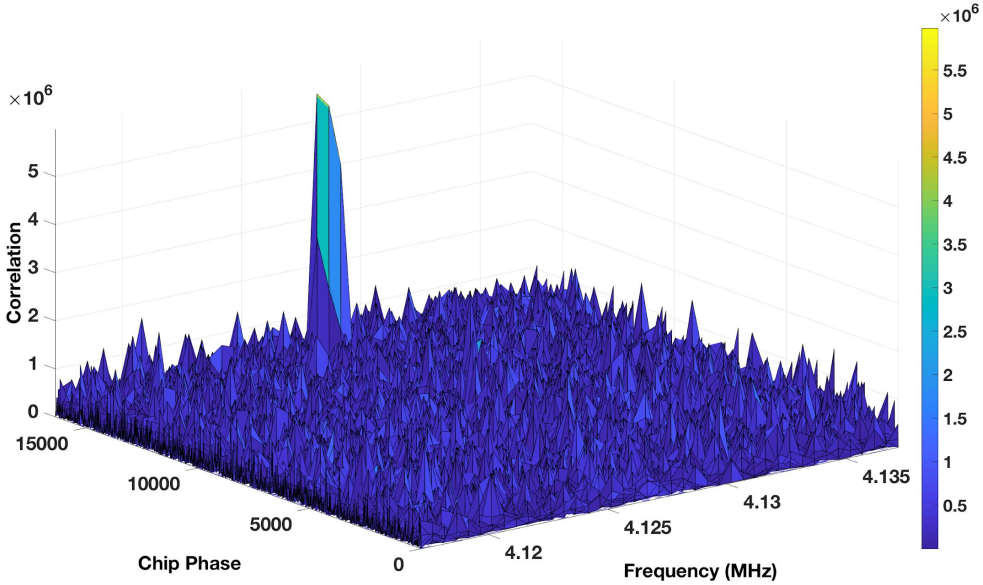
Acquisition: Serial Search

- Correlator: correlation of 1 frequency & code phase
  - Multiple correlators: NCO & ADC signals reuse, acquisition time reduces
  - parallel code phase search, serial frequency search
  - Parameterized number of correlators, number of integration cycles, etc.



Acquisition: Results and Next Steps

- FFT Search takes too much time in compile and simulation, only serial search is verified
- With real GPS data, the optimal frequency and code phase is located successfully
- Both methods are taking much more area than expected, to be solved in future

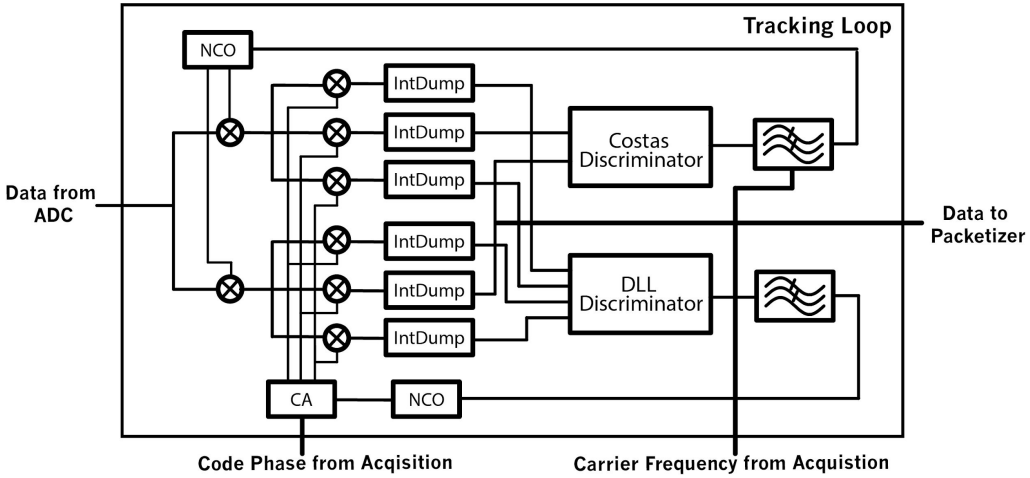


Results with 40 frequencies with 500Hz frequency steps and 2046 code phases with 8 code phase step

Tracking

Tracking: System Overview

- The tracking loop ensures that the frequency and code phase passed in from the acquisition loop stay locked
  - The code phase lock is maintained using a delay locked loop (DLL)
  - The carrier frequency and phase lock using a frequency assisted Costas loop
- Each loop has a discriminator that turns the correlator outputs into meaningful data and a loop filter that reduces the noise of the discriminator output

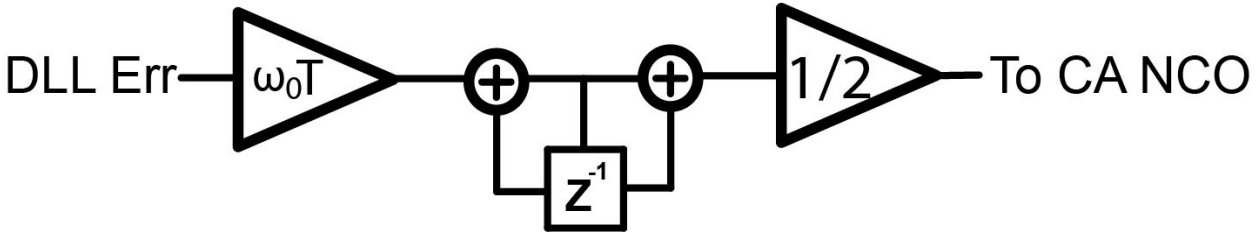


Tracking: Code Recovery

- The DLL consists of a discriminator defined by:
  - Normalized early-minus-Late envelope
    - Normalization removes amplitude sensitivity
  - Moderate computational load, no square root calculations

$$E = I_E^2 + Q_E^2$$
$$L = I_L^2 + Q_L^2$$
$$\Delta\phi_c = \frac{1}{2}\frac{E - L}{E + L}$$

- Also has a first order loop filter that drives the two code generator NCOs:
  - The discrete bilinear transform of a first order analog integrator
  - Unconditionally stable

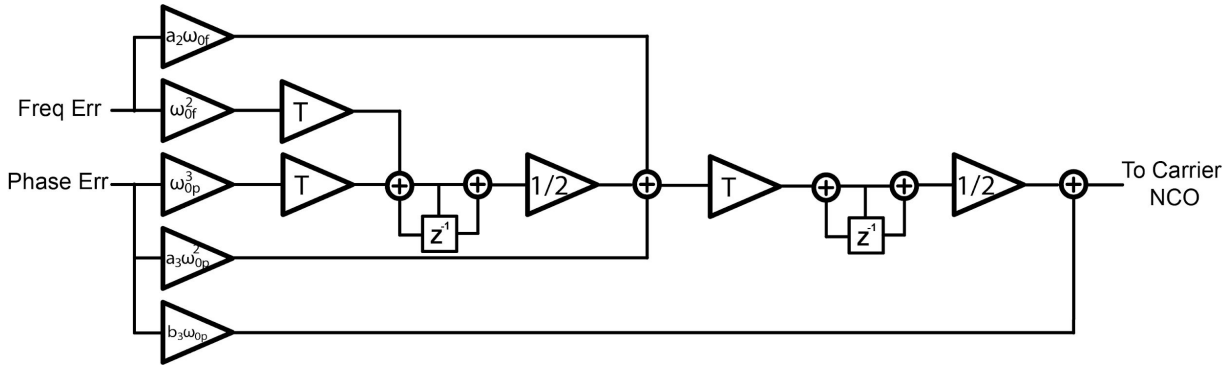


Tracking: Carrier Recovery

- The Costas loop consists of two discriminators:

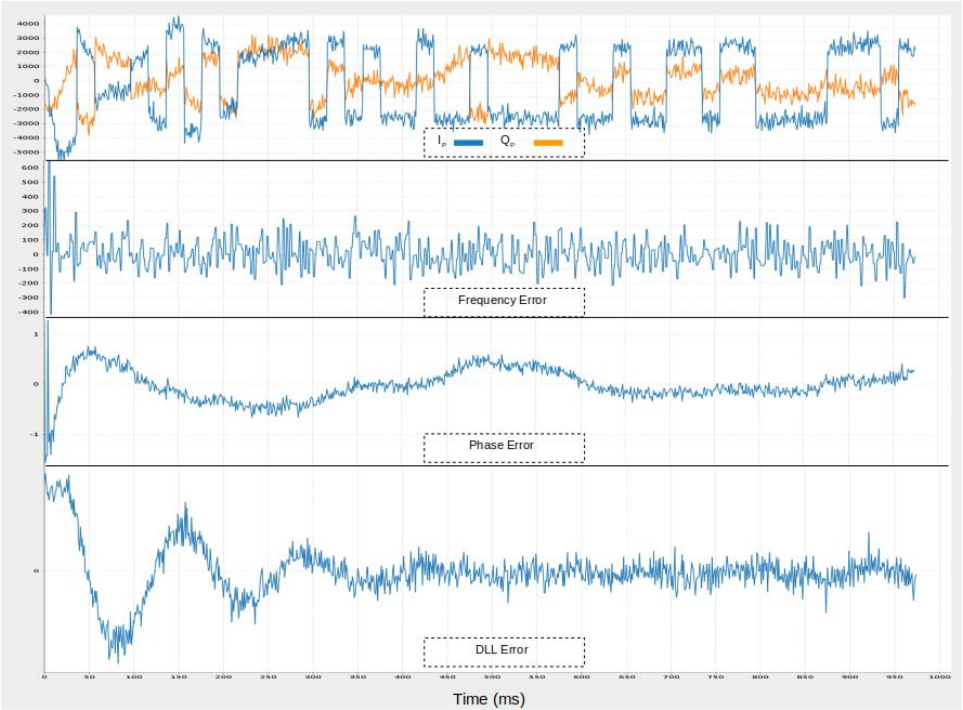
$$\phi_e = \text{ATAN}\left(\frac{Q_p}{I_p}\right) \qquad \omega_e = \frac{\text{ATAN2}\left(Q_p, I_p\right)}{\Delta t}$$

  - A bit transition invariant Costas discriminator using 2 quadrant arctan
  - A bit transition sensitive frequency discriminator using 4 quadrant arctan
    - Added a integrator-based synchronizer to align to bit flips
- The two errors are fed into a 3rd/2nd order filter that drives the carrier NCO



Tracking: Results

- Implemented all the RTL, including CORDICs, for discriminators and loop filters
- Able to track GPS satellites from the example Dataset in a Scala/RTL mixed simulation



Tracking: Next Steps

- Full RTL simulation of Loops does not yet converge due to LSB issues in the CORDICs, need to optimize sizing of Fixed Points
- Current design requires 3 Cordics per tracking channel, need to create a system to “timeshare” a single CORDIC block per channel and loop calculation
- Loop can only do 1ms integration times, need to allow the loop dynamically to dynamically switch to 2, 5, 10 and 20ms integration times
  - Implement a lookup table based coefficient setter for the loop-filters

Ranging

Ranging Overview

- The ranging code is responsible for the pipeline of hardware decoding to user position. This is the ultimate goal of the GPS receiver.
- Ranging is a three step process:
  - Extract SV parameters and apply scaling/unit conversions
  - Compute “GPS time” and SV position
  - Compute user position

Hardware Extraction - Packetizer

- Bit-domain decoding of data
  - Basic unit of data: 1 word = 30 bits
  - 10 words form a subframe, 5 subframes form a frame
  - Positioning data found in first 3 subframes
- Three submodules in series - parser, parity checker, param extractor

TABLE 5.5 Parity Encoding Equations

$$
\begin{aligned}
d_1 &= D_1 \oplus D_{30}^* \\
d_2 &= D_2 \oplus D_{30}^* \\
d_3 &= D_3 \oplus D_{30}^* \\
&\vdots \\
d_{24} &= D_{24} \oplus D_{30}^* \\
D_{25} &= D_{29}^* \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{17} \oplus d_{18} \\
&\quad \oplus d_{20} \oplus d_{23} \\
D_{26} &= D_{30}^* \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{18} \oplus d_{19} \\
&\quad \oplus d_{21} \oplus d_{24} \\
D_{27} &= D_{29}^* \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_8 \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{19} \\
&\quad \oplus d_{20} \oplus d_{22} \\
D_{28} &= D_{30}^* \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{20} \\
&\quad \oplus d_{21} \oplus d_{23} \\
D_{29} &= D_{30}^* \oplus d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_9 \oplus d_{10} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{18} \\
&\quad \oplus d_{21} \oplus d_{22} \oplus d_{24} \\
D_{30} &= D_{29}^* \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{11} \oplus d_{13} \oplus d_{15} \oplus d_{19} \oplus d_{22} \\
&\quad \oplus d_{23} \oplus d_{24}
\end{aligned}
$$

Parity check equations (*Fundamentals of Global Positioning System Receivers: A Hardware Approach*)

Hardware Extraction - Packetizer (cont.)

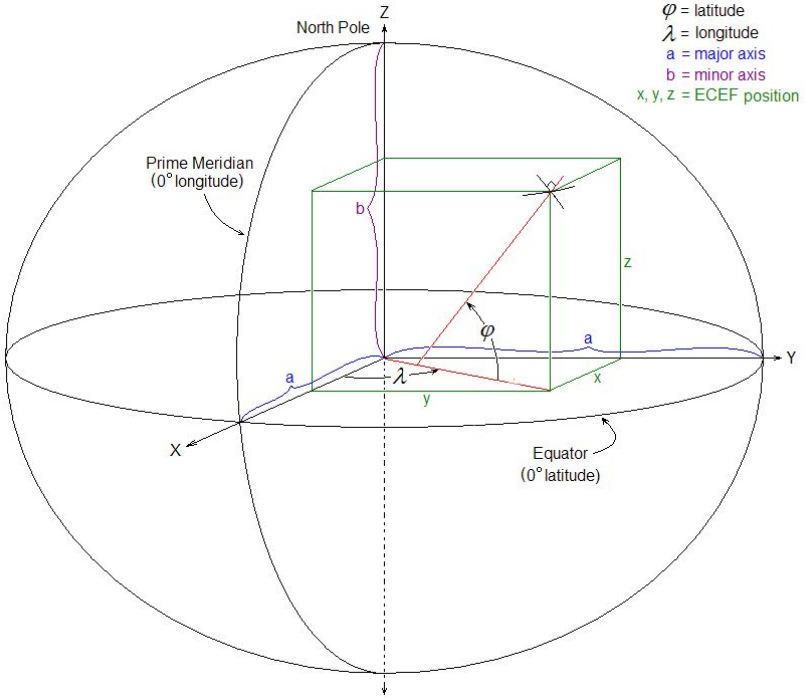
- Parser
  - Detects 8-bit GPS preamble that indicates the start of a subframe
  - Reads 300 bits into a buffer and passes to next stage
- Parity checker
  - Each word has 24 data bits and 6 parity bits
  - Parity bits are calculated with certain XOR patterns of the 24 data bits and the 2 last bits of the previous word
  - Parity check must pass on all words of a subframe for data to be valid
- Param extractor
  - Identifies subframe and extracts relevant parameters for ranging algorithm
  - This submodule was added later, as the bit manipulations it requires were inefficient in software

Hardware Extraction - Packetizer (cont.)

- Firmware-hardware interfacing done through a regmap
  - Read UInts/SInts from Chisel, convert data types and apply scaling in hardware
  - Package as a struct to be passed to ranging algorithm
- Different memory block per tracking channel

SV Position Calculation

- "GPS time" is a local replica of the SV clock
- SV ephemeris data contains info necessary to compute this time delay
- SV Position is computed using an established set of equations that include correction for motion and other non-idealities



ECEF (Earth-Centered, Earth-Fixed) Coordinates
Image source: https://en.wikipedia.org/wiki/ECEF

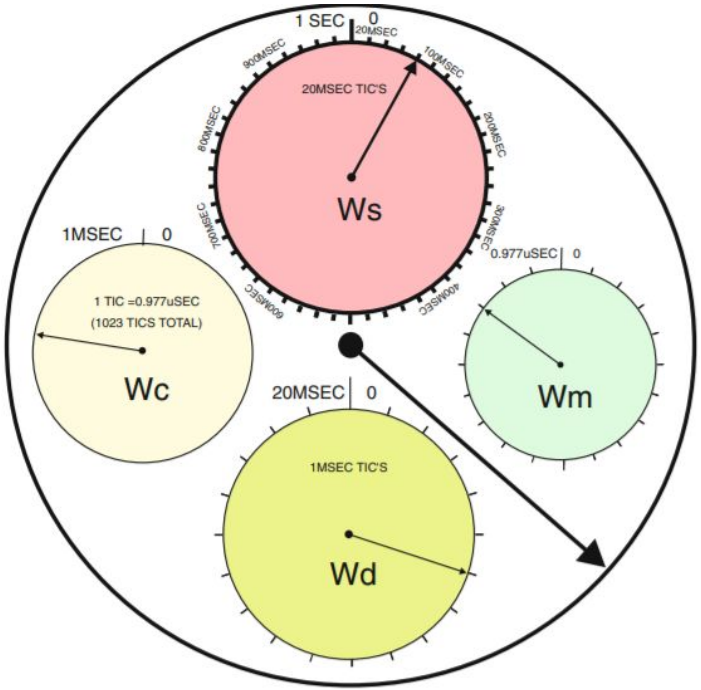
BWRC

SV Position Calculation (Cont.)

- After parameter extraction, local replica clock is created and SV position is computed
- These results are stored in a struct
- After four sets of SV data are populated, they are sent to the user position calculation

Receiver Position Calculation

- Inputs:
  - 4 SV Positions
  - Signal transmit and receive times
- Outputs:
  - Receiver position
  - Receiver time bias
- Tracking loop is a model of the SV signal at transmit
- Pseudorange
- Iterative calculation



Doberstein. Fundamentals of GPS Receivers, Springer. 2012.

BWRC

Ranging Conclusion and Next Steps

- The skeleton for computation is complete, but is not fully integrated with hardware due to incomplete datapath
- Using decoded data from jks.com/gps/gps.html we were able to verify that the SV position calculation is correct
- Decoding parameters and recording times from tracking loop measurements

  BWRC Location (km):
  - X: -2641.466
  - Y: -4262.826
  - Z: 3894.033

Integration

Integration: Results and Next Steps

- The tracking channel has been connected to Rocket using a register mapped approach

Next Steps:

- The next integration component is to connect the acquisition loop with the tracking loop
  - The current plan is to connect the two using RocketChip
  - Cannot be done until after the acquisition area has been optimized
- Ranging code also needs to be tested with the tracking loop hardware and Rocket

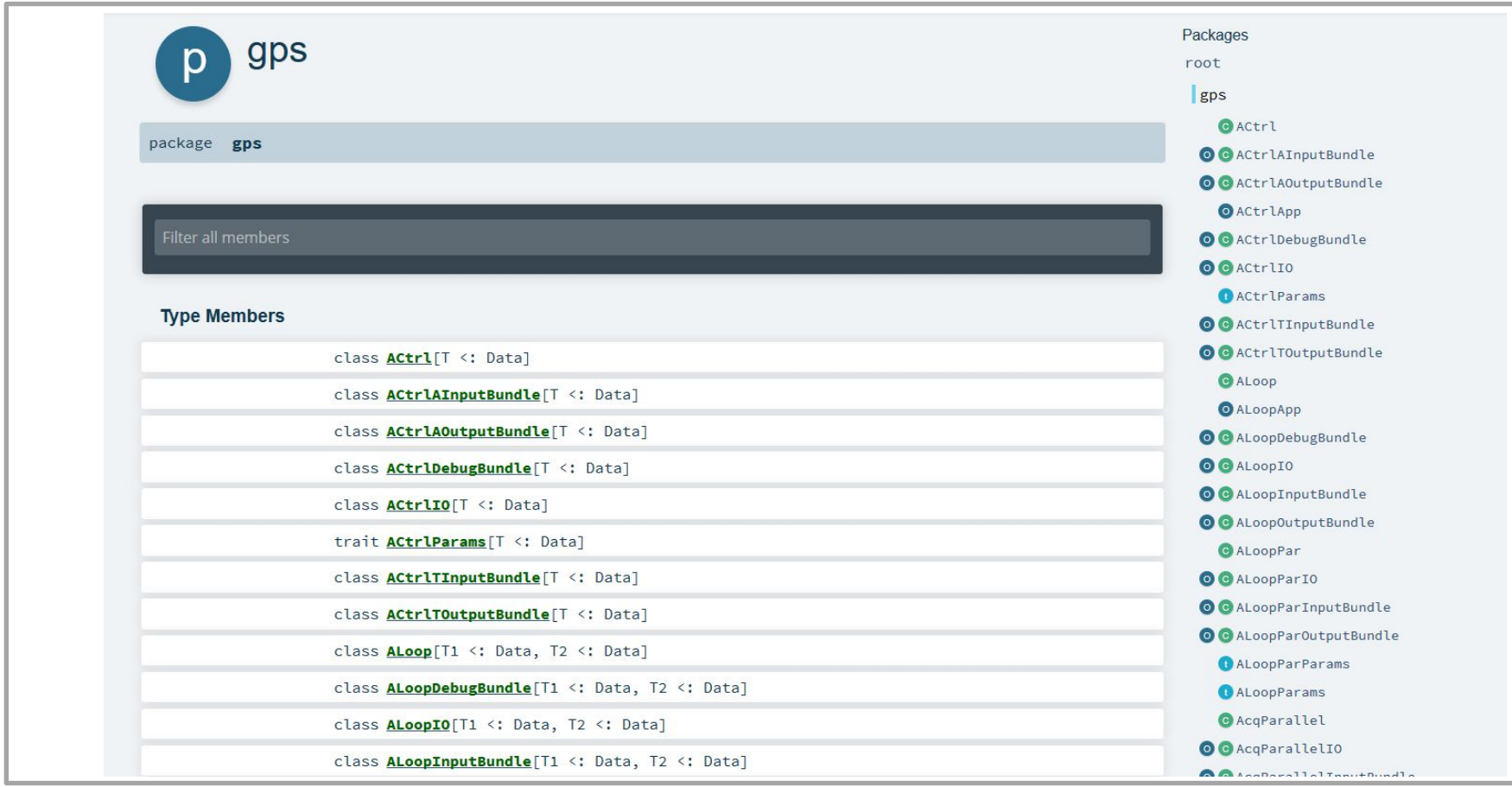
Conclusion

Conclusion

- CHISEL was a great tool
  - Able to generate variants on submodules - reusability!
  - Scala was conducive to test automation
  - Ability to straightforwardly integrate software/hardware
  - Top-level integration workflow was very different from other HDLs - took some getting used to
- DspTools is critical for CHISEL DSP applications
  - FixedPoint!
- Connecting with RocketChip gives instant ability to test system as an SoC

Documentation

Scala Docs



p gps

package **gps**

Filter all members

Type Members

class **ACtrl**[T <: Data]

class **ACtrlAInputBundle**[T <: Data]

class **ACtrlAOutputBundle**[T <: Data]

class **ACtrlDebugBundle**[T <: Data]

class **ACtrlIO**[T <: Data]

trait **ACtrlParams**[T <: Data]

class **ACtrlTInputBundle**[T <: Data]

class **ACtrlTOutputBundle**[T <: Data]

class **ALoop**[T1 <: Data, T2 <: Data]

class **ALoopDebugBundle**[T1 <: Data, T2 <: Data]

class **ALoopIO**[T1 <: Data, T2 <: Data]

class **ALoopInputBundle**[T1 <: Data, T2 <: Data]

Packages

root

gps

ACtrl
ACtrlAInputBundle
ACtrlAOutputBundle
ACtrlApp
ACtrlDebugBundle
ACtrlIO
ACtrlParams
ACtrlTInputBundle
ACtrlTOutputBundle
ALoop
ALoopApp
ALoopDebugBundle
ALoopIO
ALoopInputBundle
ALoopOutputBundle
ALoopPar
ALoopParIO
ALoopParInputBundle
ALoopParOutputBundle
ALoopParParams
ALoopParams
AcqParallel
AcqParallelIO

BWRC

Scala Docs

gps

CA

```
class CA extends Module
```

A GPS L1 C/A PRN code generator module. IO:

satellite: Input(UInt), an int between 1 and 32 that selects which satellite's CA code to generate.

fco: Input(SInt), an NCO that drives the update frequency of the CA code

fco2x: Input(SInt), an NCO at 2x the frequency of fco that drives the update of the CA code

early: Output(SInt), The CA code, updated at zero-crossings of fco

late: Output(SInt), see CACodeShiftReg

punctual: Output(SInt), see CACodeShiftReg

done: Output(Bool), true for the cycle that a 1023-length CA code has fully finished

currIndex: Output(UInt), the current index of the 1023 length CA code being outputted

Linear Supertypes

Filter all members

Instance Constructors

```
new CA(params: CAParams)
```
create a CA code generator module

Value Members