Robot Waypoint Planner: Project Presentation

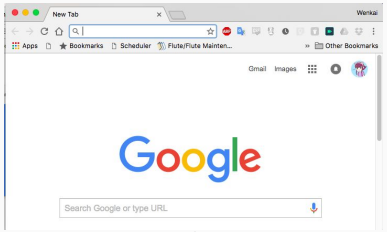
Wenkai Qin
Jack Yang

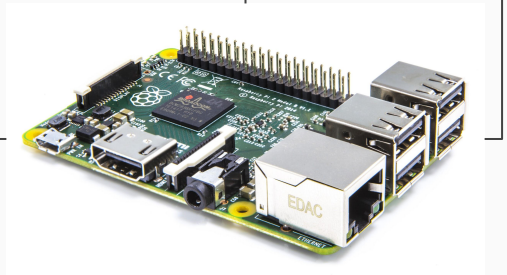
Project Goal

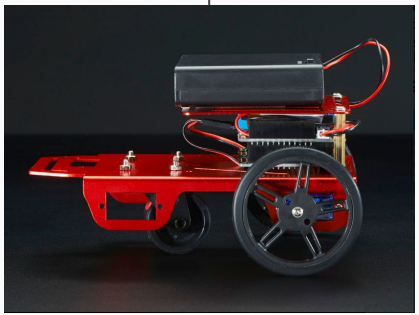
We want to build a robot waypoint planner that has...

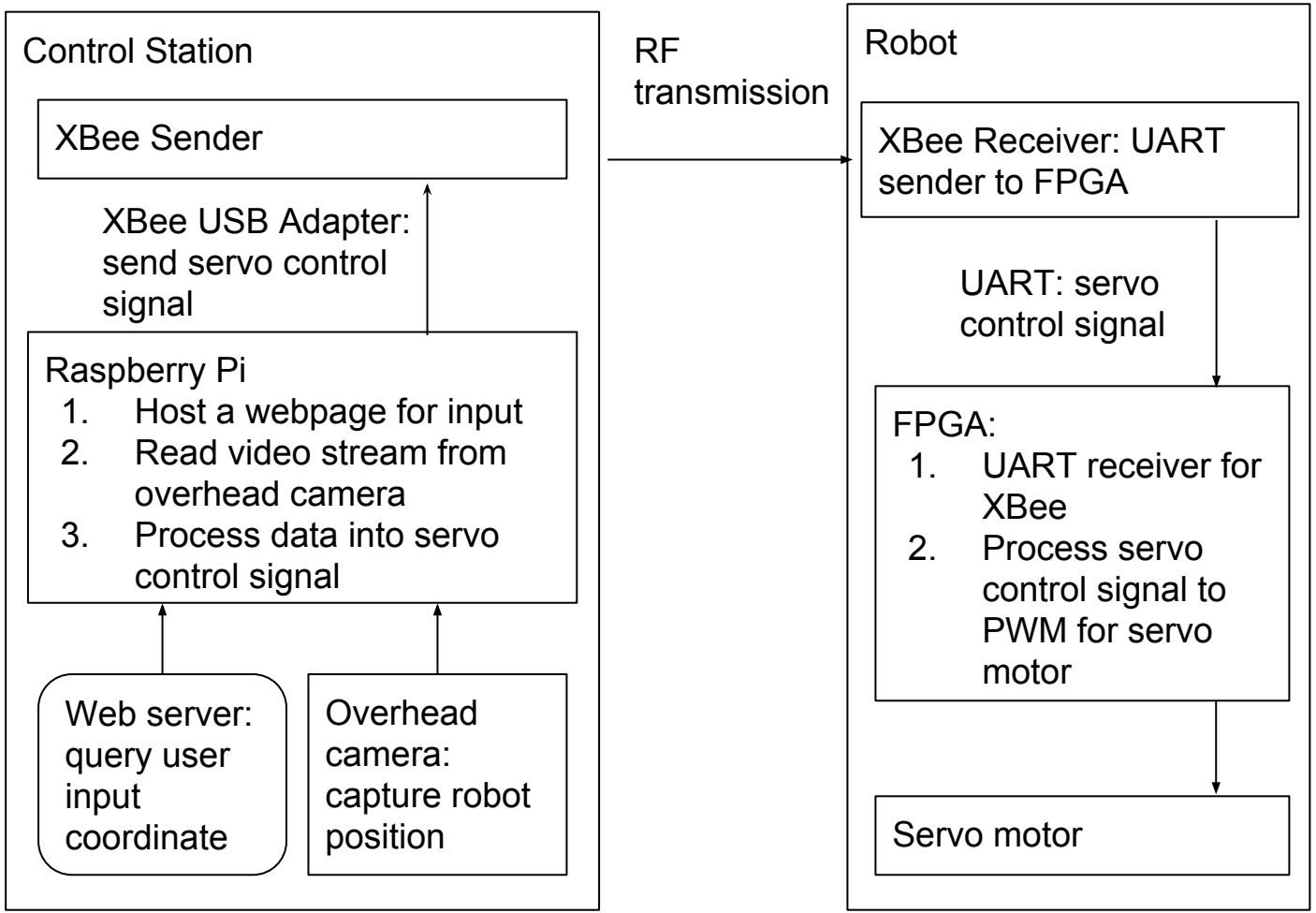
- Closed-loop control
- Wireless communication

Project Overview





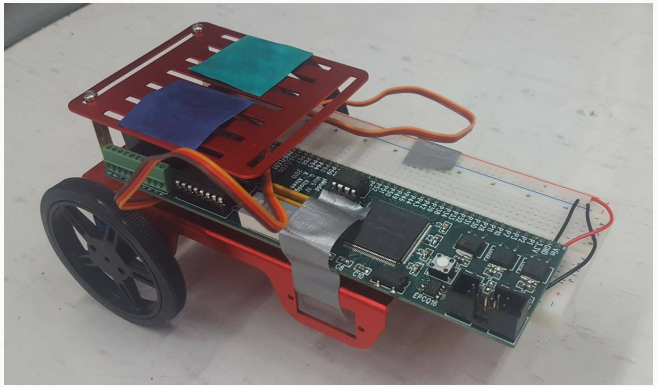



Control Station
XBee Sender
XBee USB Adapter: send servo control signal
Raspberry Pi
1. Host a webpage for input
2. Read video stream from overhead camera
3. Process data into servo control signal
Web server: query user input coordinate
Overhead camera: capture robot position
RF transmission
Robot
XBee Receiver: UART sender to FPGA
UART: servo control signal
FPGA:
1. UART receiver for XBee
2. Process servo control signal to PWM for servo motor
Servo motor

Project Overview

On the robot, we have

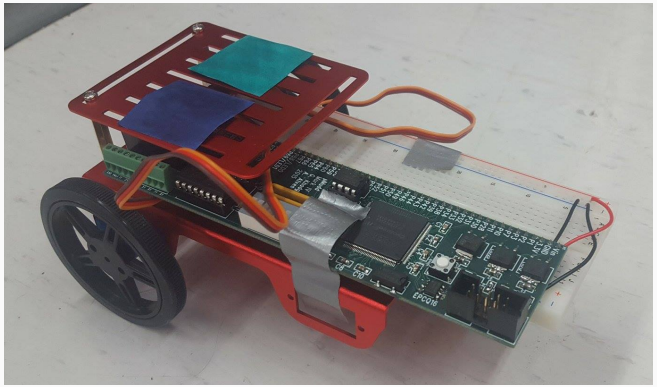
- FPGA:
  - Servo control using PWM
  - UART interface with onboard Xbee
- XBee
  - Receive control signal
- Li-Po battery, servo, voltage regulator…



Project Overview

On the robot, we have

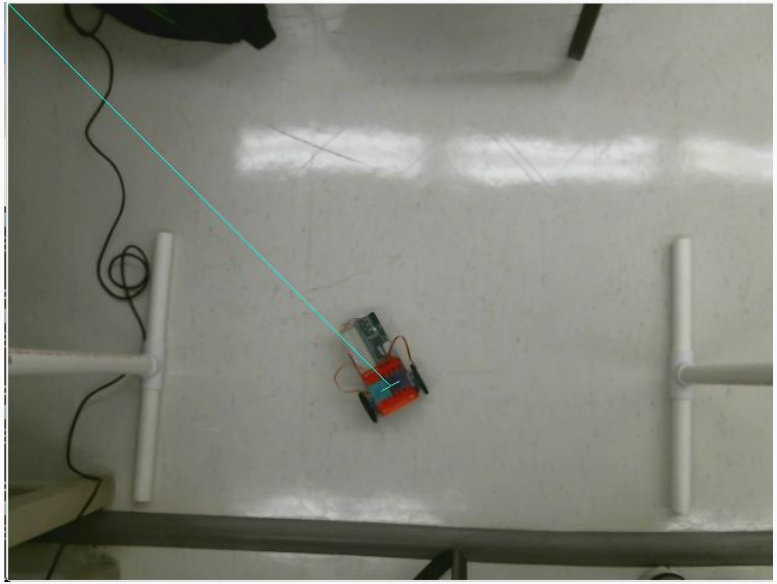
- FPGA:
  - Servo control using PWM
  - UART interface with onboard Xbee (Done)
- XBee
  - Receive control signal (Done)
- Li-Po battery, servo, voltage regulator… (Done)



Project Overview

For control station, we have

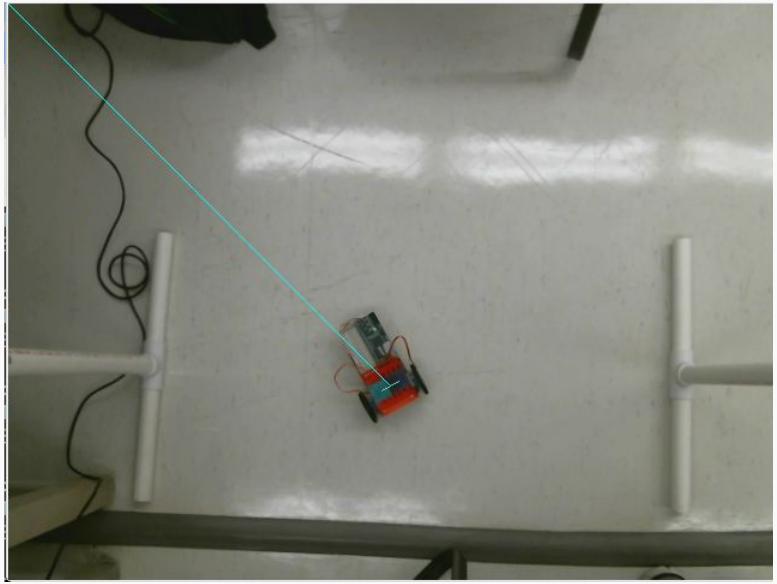
- Raspberry Pi
  - Web interface: user input
  - Overhead camera: track robot position & orientation
  - Robot controller
  - USB link to XBee
- XBee
  - Sends control signal to robot



Project Overview

For control station, we have

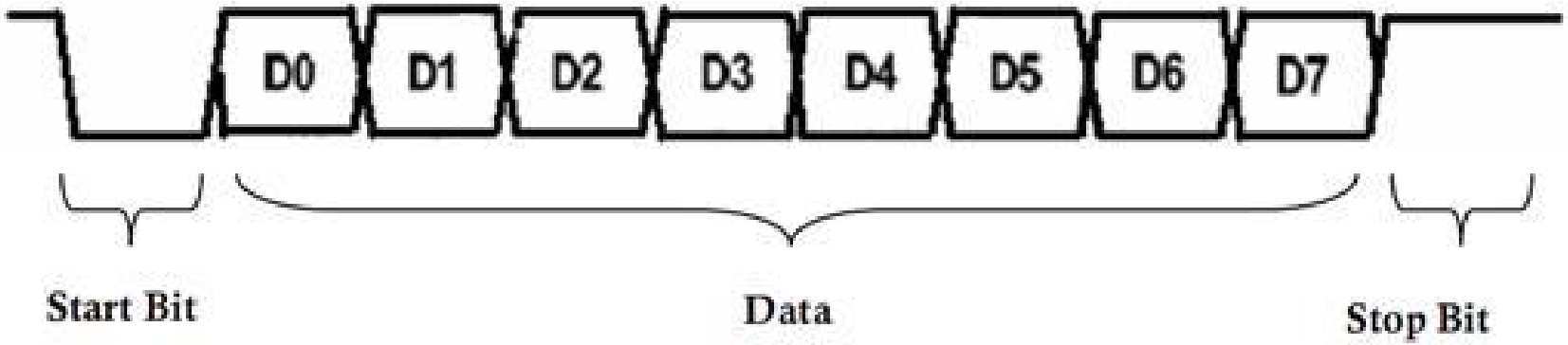
- Raspberry Pi
  - Web interface: user input
  - Overhead camera: track robot position & orientation (Done)
  - Robot controller (In progress)
  - USB link to XBee (Done)
- XBee
  - Sends control signal to robot (Done)



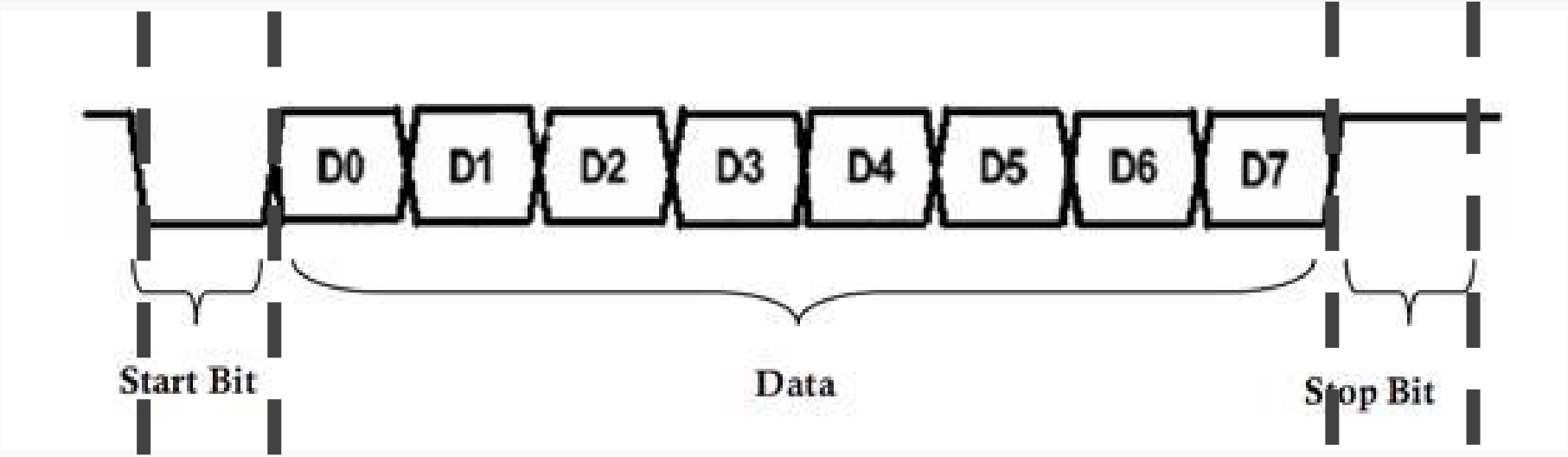
Deep Dive: UART Implementation

- Asynchronous, full duplex
- For us, only **receiving data** from XBee
- Clock generation
- Data integrity

Timing Diagram



Timing Diagram



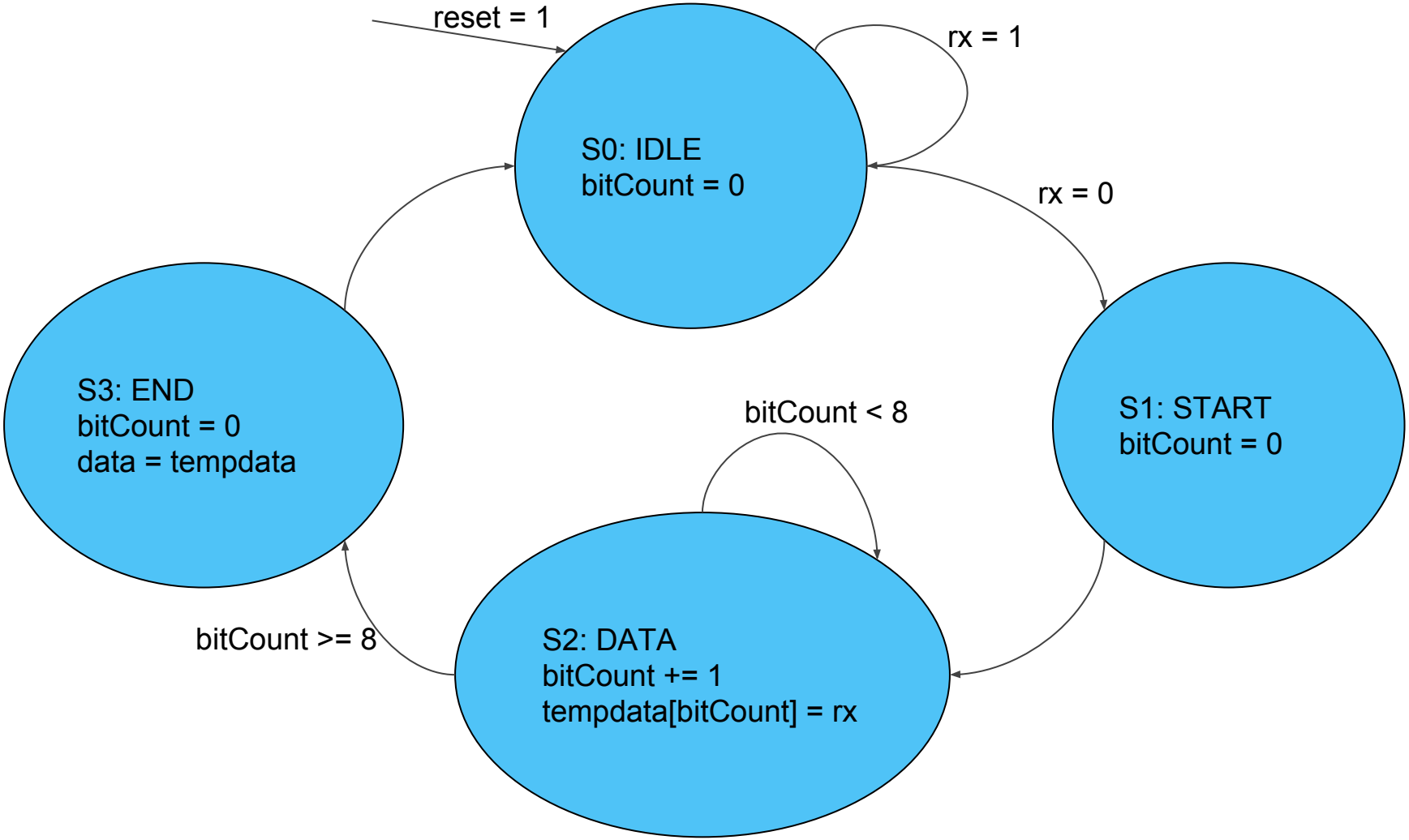
S0: IDLE

S1: START

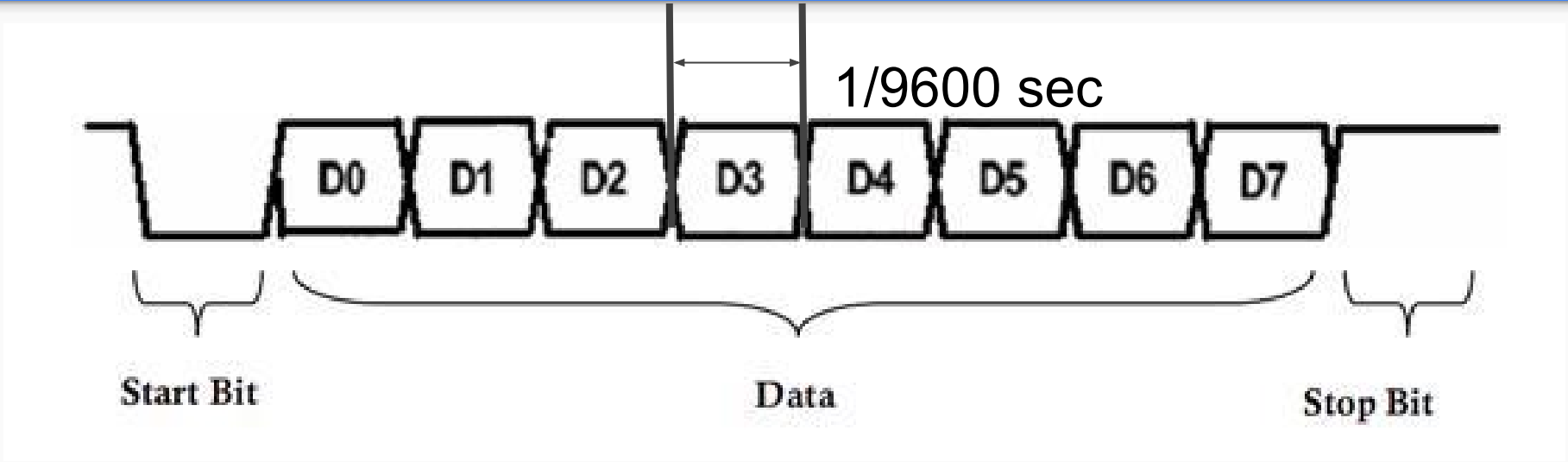
S2: DATA

S3: END

S0: IDLE


reset = 1
rx = 1
S0: IDLE
bitCount = 0
rx = 0
S1: START
bitCount = 0
bitCount < 8
S2: DATA
bitCount += 1
tempdata[bitCount] = rx
bitCount >= 8
S3: END
bitCount = 0
data = tempdata

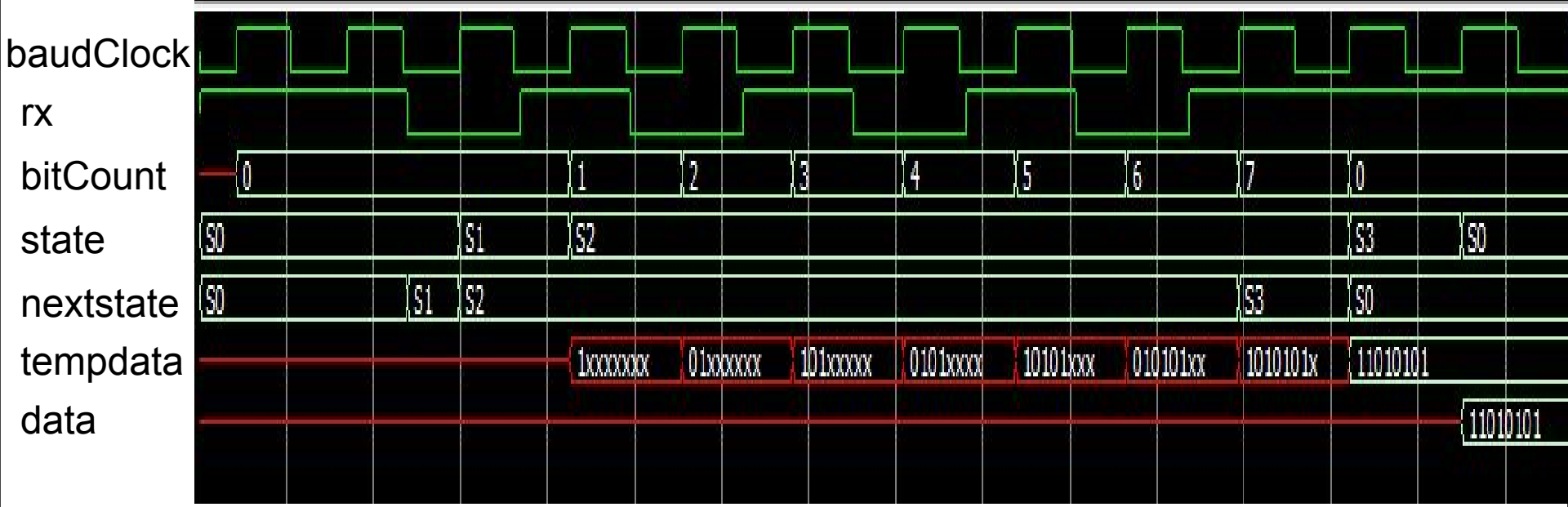
Baud Rate: 9600 Hz

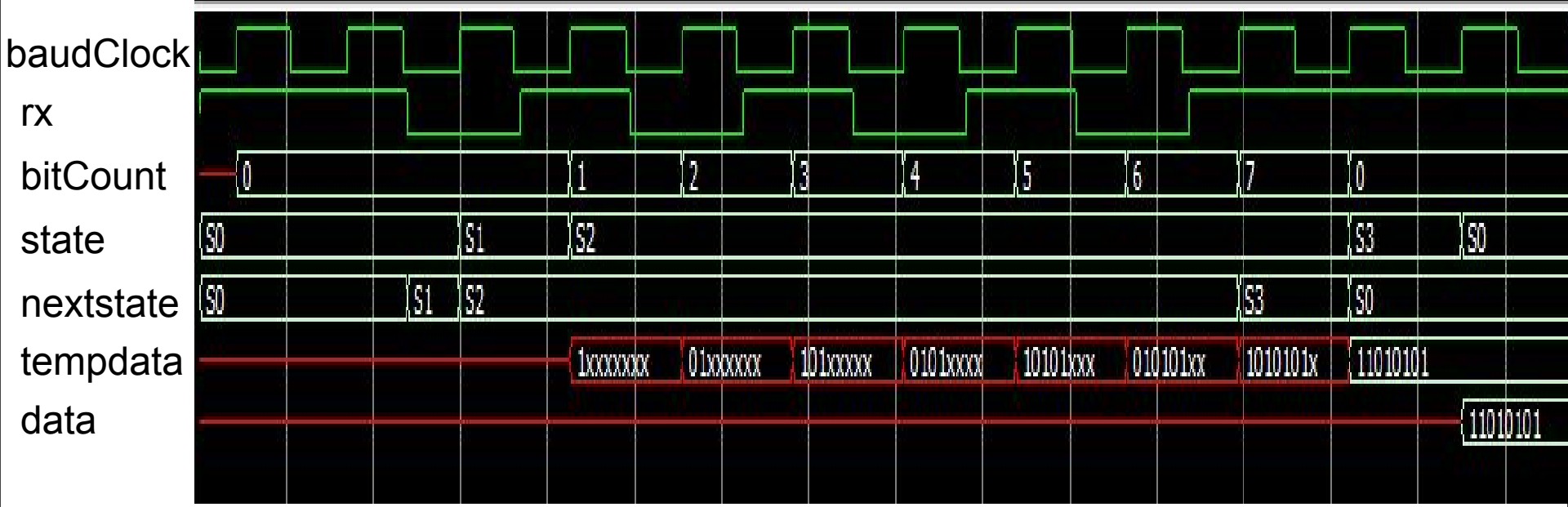


How to generate a clock of 9600 Hz?

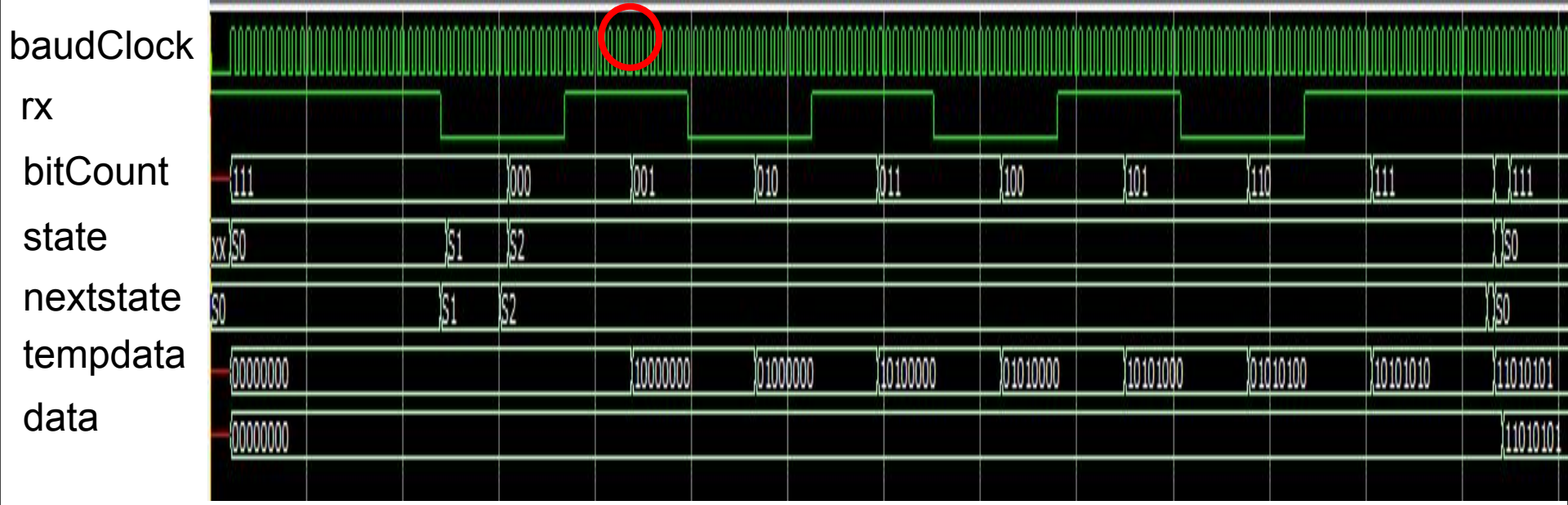
```
logic [17:0] counter;                          // Max value = 2^18 - 1
logic [17:0] increment = 18'b111111;           // Increment = 63
always_ff@(posedge clk, posedge reset)  // Clock on FPGA is 40 MHz
      if(reset) counter <= 18'b0;
      else counter <= counter + increment; //
assign bck = counter[17];                      // 40MHz * 63/(2^18-1) = 9613.04Hz

                                    // which is 0.136% away from the desired value of
                                  9600Hz. This error is within the tolerance of UART.
```


baudClock
rx
bitCount
state
nextstate
tempdata
data
0
1
2
3
4
5
6
7
0
S0
S1
S2
S3
S0
S0
S1
S2
S3
S0
1xxxxxxx
01xxxxxx
101xxxxx
0101xxxx
10101xxx
010101xx
1010101x
11010101
11010101



- Sample once at the rising edge of baudClock is not a reliable method, because rx and baudClock will not perfectly align.
- Data subject to drifting and jittering.



Oversampling:

- Generate baudClock with frequency of 16*baud rate.
- Sample 5 times in the middle of rx.
- Check if all 5 samples are consistent. If not, discard the data.

Questions?

- How to avoid the clock drift and improve data quality?
- How should we encode the 8-bit data packet to control two servos?
- How to extract camera feed to web interface while processing camera feed?
- Suggestions on webpage design, robot control, etc..