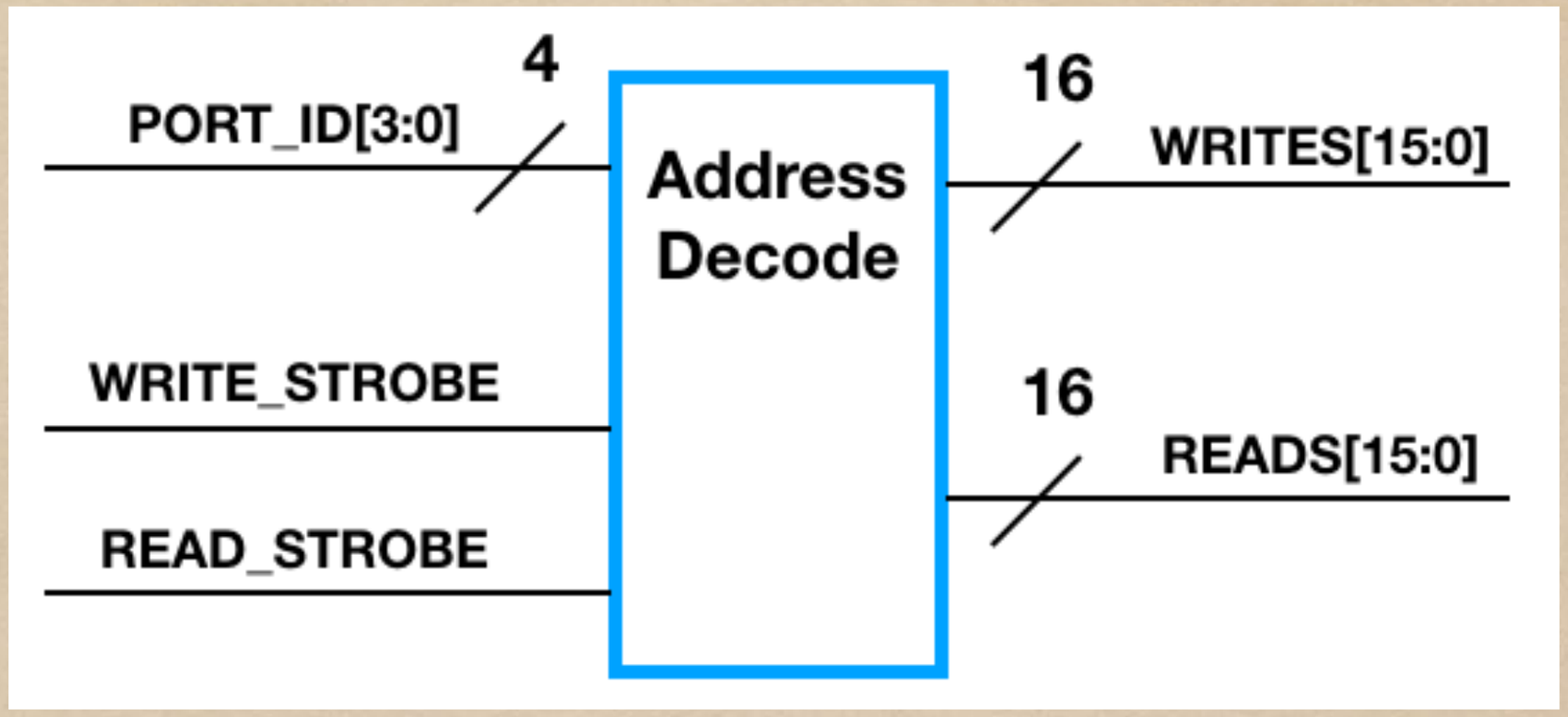
Full UART Transmitter & Receiver

CSULB
CECS 460

Full UART

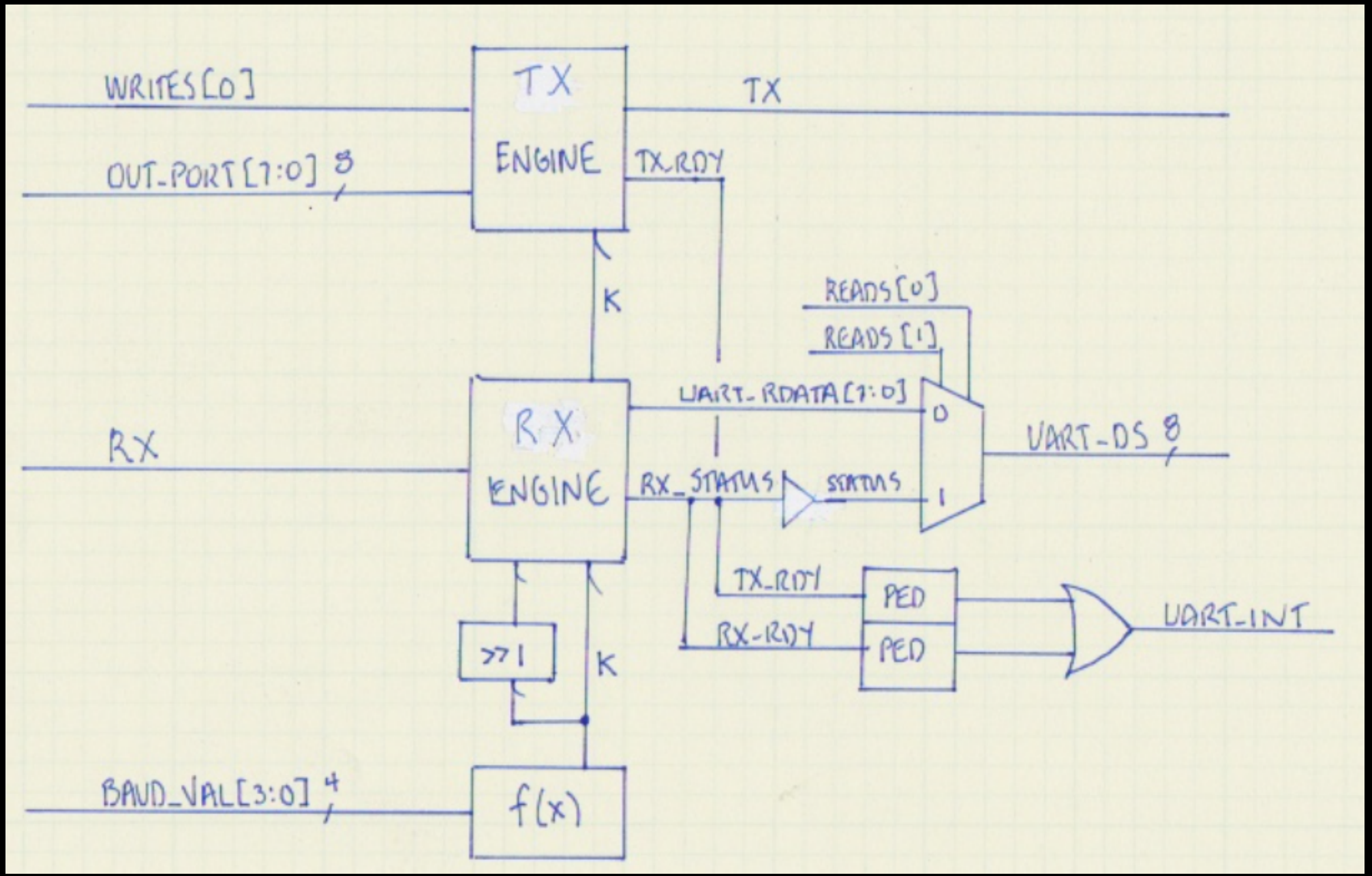
- In both class lectures and posted videos the full UART has been presented
- In this project you will create the full UART as an IP in your design.
- The simple block diagram will document what your design should include at this time
- The following diagrams show the architecture of both the **data path** and the **control** portions of the receive engine

Address Decoder



- The tramelblaze needs to access data from a variety of sources within the SOC
- To generate specific enables the design requires an address decoder to generate write and read enables
- This generates 16 write enables and 16 read enables
- If you need more enables just add more PORT_ID bits

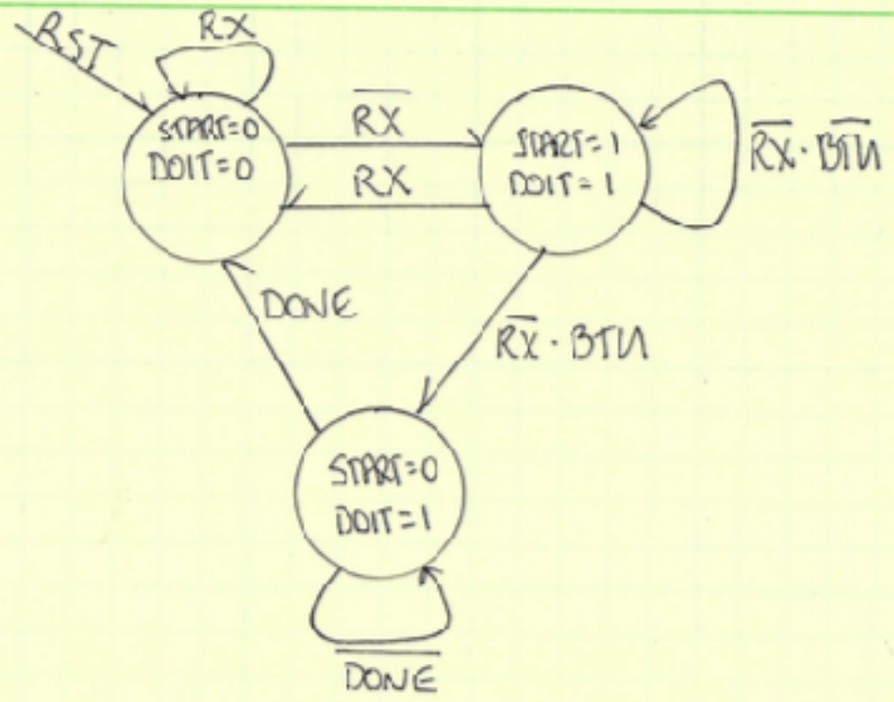
UART Top Level

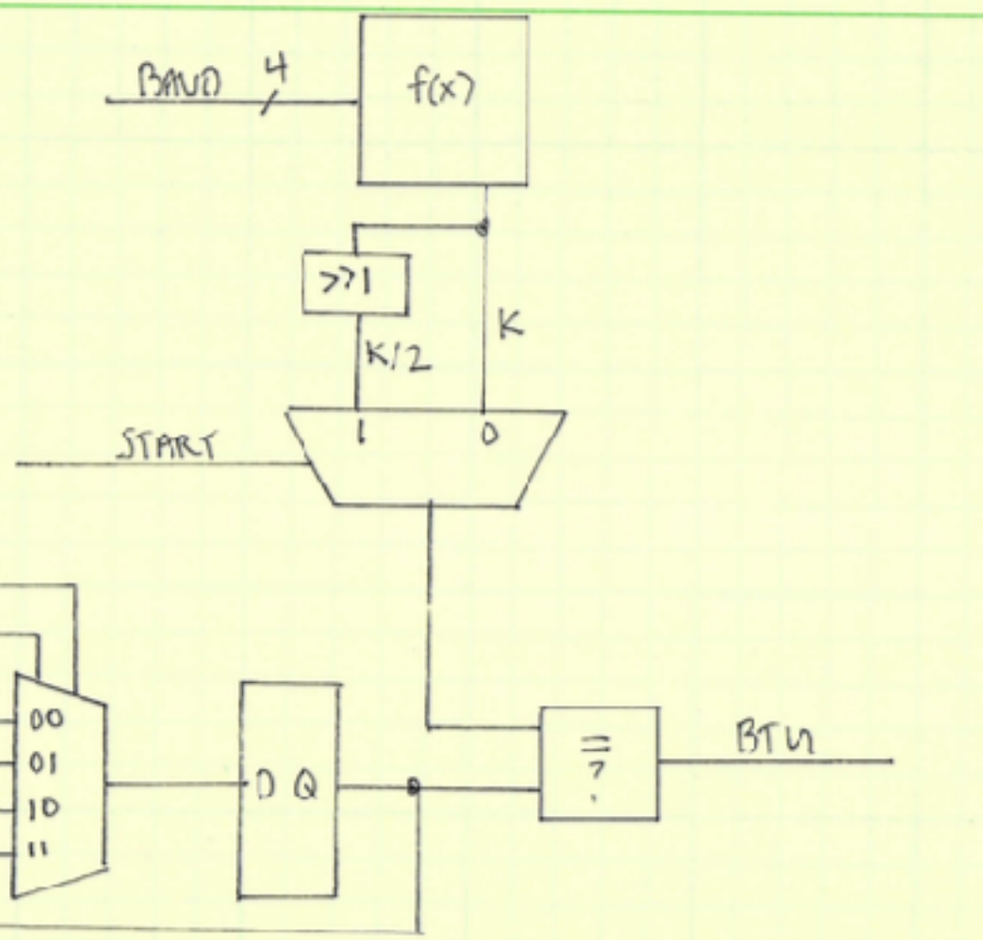


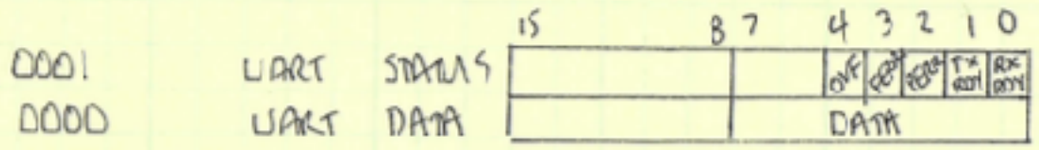
UART Top Level

- UART top level will have the TX Engine and the RX Engine along with supplemental logic
- Supplemental logic
  - Baud rate decoder generating k and k/2
  - Multiplexer selecting between the UART data and the UART status
  - The TX_RDY and RX_RDY signals need to be run through PED circuits to generate the UART_INTERUPT to the tramelblaze

Receive Engine Control







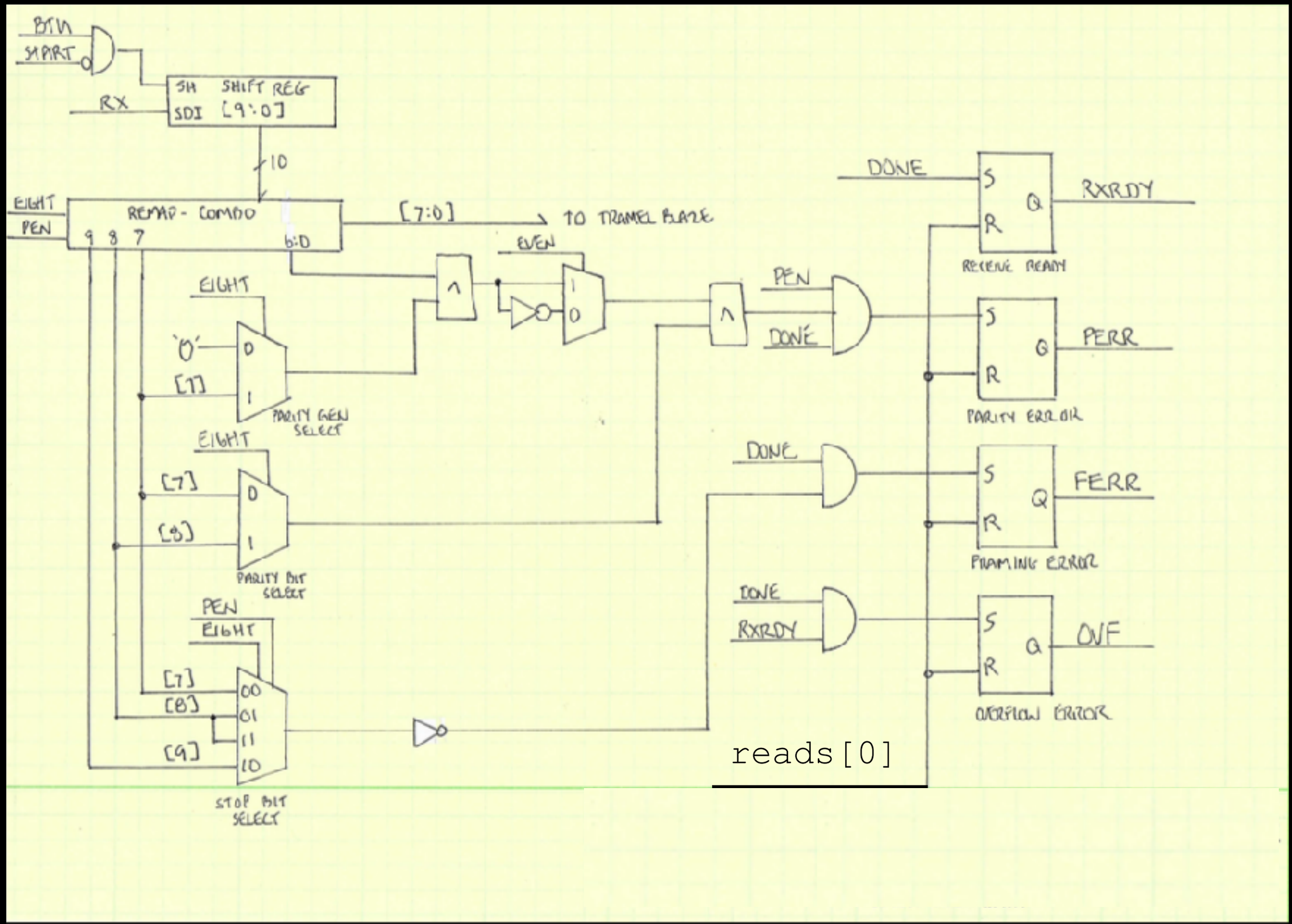
Receive Engine Control

- The goal of the receive engine is to synchronize the data collection with the TX communication
- The receive engine is always polling the RX line looking for a high to low transition indicating the arrival of the START bit
- A simple state machine ensures that the START bit remains active (0) until the mid-bit time at which time the data collection will proceed at a bit time until all bits received
- The state machine outputs are START and DOIT. START indicated that the receive engine is looking for the START bit. DOIT indicates that the receive engine is currently processing data
- The BTU signal is utilized to detect the end of the START bit (k/2) first and then to perform the collection of the DATA bits (k)

Receive Engine Status

- There are two addresses dedicated to the UART: 0000 (UART Data) and 0001 (UART Status)
- When TramelBlaze writes address 0000 the transmit engine will receive the data byte. When TramelBlaze reads address 0000 the receive engine will respond with the received data
- When TramelBlaze reads address 0001 it will receive the status collected by the UART receive engine along with the two ready bits (TX_RDY, RX_RDY)
- PERR indicates that a Parity Error has been detected
- FERR indicates sync was lost and is detected when the STOP bit is not found at the proper time
- OVF indicates that another byte is being received before the TramelBlaze has read the previous byte

Receive Engine Data Path



Receive Engine Data Path

- UART top level will have the TX Engine and the RX Engine along with supplemental logic
- Supplemental logic
  - Baud rate decoder generating k and k/2
  - Multiplexer selecting between the UART data and the UART status
  - The TX_RDY and RX_RDY signals need to be run through PED circuits to generate the UART_INTERUPT to the tramelblaze

Receive Engine Data Collection

- The BTU signal is used to sample to data but when the START bit is active there should be no data collection
- Once the desired number of bits has been received (based on configuration) the data must be analyzed to detect errors: PER, FERR, OVF
- The REMAP function is responsible for “right justification” of the received data - so we can assume the bit positions for processing

Verification

- Verifying the receive engine proves to be problematic in that the designer must generate the RX data with the correct format and timing
- One approach is to create a simple stimulus in the test fixture which will transmit a byte with the correct timing and format
- A second approach is to utilize the verification environment created for the transmit engine
- Here the TX signal is captured and saved into to a file to be read into a memory and applied to the RX engine

Test Fixture Stimulus

- Simulation of designs before programming the FPGA remains a necessary step if you want to produce a working design in a time efficient manner
- The transmit engine was “self sufficient” in that once a clock was supplied along with a reset the machine would run by itself
- The receive engine now is the initiator of activity within our design and therefore there needs to be an input stimulus provided on the RX signal
- What follows is a technique whereby you can generate your RX stimulus by using a successful TX simulation

Constructs for Test Fixture - Creating the Vectors

```
// variables for file
integer      f;              // file reference
reg          flip;           // flag to designate write period
time         t;              // variable to control write period

//
```

```
  initial begin
       f = $fopen("output.txt");      // open file to output stimulus
       flip = 0;                      // initialize the write flag
]
```

```
 always #t
       begin
       flip = !flip;                  // toggle to observe when writing
       $fwrite(f,tx," ");             // write value of tx to file
       end
```

```
#100
reset = 1'b0;                         // deassert reset
#50_000_000                           // run for adequate amount of time
$fclose(f);                           // close the file
$finish;                              // end the simulation
end
```

Constructs for Test Fixture - Applying the Vectors

```
// Example program demonstrating reading vectors for rx simulation
// output.txt file created during transmit simulation
// Vectors read into memory for application during receive simulation
// cecs 460 spring 2017 john tramel

`timescale 1ns/1ns

module ReadIt();

reg [0:0] mem [0:999_999];          // declare memory for vector
integer   i;
reg       rx;                       // applying value to rx signal


initial begin
  $readmemb("output.txt",mem);      // read created file into vector memory
  for (i=0; i<1_000_000; i = i + 1) // iterate to apply vector stimulus
     begin
     #100                           // delay should be same as when collected
     rx = mem[i];                   // apply the stimulus to rx signal
     end
  end

endmodule
```

- The time interval for collecting and applying the serial data should be about 10x. So if the bit time is 1.085 us then the sample interval should be around 108 ns (doesn't have to be exact)

Required Software

- The RX engine is driven by the TramelBlaze software
- The following will document what software should be developed for this project

Required Software

- Continue to blink the LEDs while you are idling in the **main** loop
- You should display a banner when starting out of reset and then you should provide a prompt
- When a key is pressed on the terminal a byte will be sent to the receive engine via the RX signal (notification is through RX_RDY) which will cause interrupt - you must then read the status to determine the source of the interrupt (TX or RX)
- You should process the characters received according to the following:
  - A **<CR>** should send the cursor to the start of the next line and should refresh the prompt
  - A **<BS>** should erase the character in front of the cursor - care must be taken to not erase the prompt
  - An "*****" will result in the outputting of your home town followed by a newline and a prompt
  - An "**@**" will result in the outputting of the number of characters received followed by a newline and a prompt
  - All other characters should be echoed on the display up to the 40th character - when the 40th character has been echoed then a newline and a prompt should be issued

Overview

- The project is due March 28
- The documentation is expected to be in the format compatible with the chip specification format
- You should use the recommended switches and status bit locations