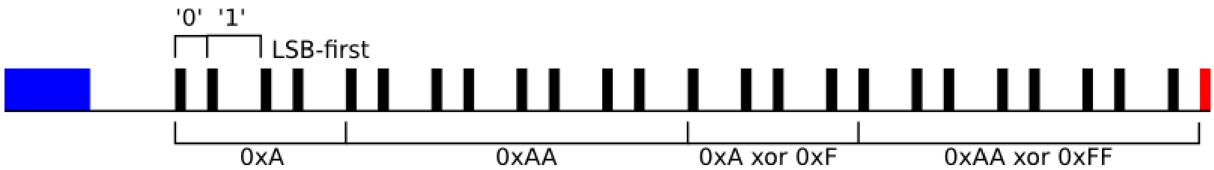
IPLoS Packet Receiving Strategy



There are two possible strategies to receive an IPLOS packet: Polling at a regular interval and interrupt-driven reception. Polling requires the controller to sample the IR receive line with a period of 500µs (the length of the *on*-signal burst), or less (multiplied by powers of one-half). This method is noise-resistant as the probability of sampling noise is small. Even if it did sample the noise, e.g. sampling a low on the line only 500µs after an *on*-signal burst, which is invalid, the programmer can perform sanity check on the length of the latest received signal section. Sampling at a rate of 500µs may cause the receiver to lose packets due to variations in signal bursts. Even with higher sampling rate the receiver will have to account for slight variations in signal section lengths.

Interrupt-driven reception requires the IR line to be connected to an external interrupt pin that triggers on signal edges, and use a timer module to calculate the length of the signal sections. This allows the controller to work on other parts of the system and the ISR to be short; but this method will capture, or at least trigger an interrupt when there is noise on the line. The programmer will have to perform (most likely extensive) checks to make sure that noise is filtered when the device is receiving a packet.

POLLING STRATEGY

Below is an example state machine to sample an IPLoS data packet from an I/O line, with sampling rate of 500µs (the controller checks the Input line every 500µs).

REGISTERS

Input:	An input pin on the controller that is pulled low when the IR receiver senses an IR signal
Count:	Interrupt counter to measure time intervals
Data:	A vector that stores the received data (24-bits)
Index:	An index into the data vector where the packet is temporarily stored
DataReady:	A flag that is used by the main program loop to determine whether a new packet is ready to be processed. The state machine will wait on this signal when a packet is read until it is cleared by the main loop.

STATES

SCAN (RESET): Scan the IR line and move to the next state if it goes low

```
Input == high → SCAN

Input == low → Count := 0, CHECK_START_LOW
```

CHECK_START_LOW: Make sure that the initial *on* burst is +/- 2000µs.

```
Count < 3 && Input == high → SCAN   // invalid, burst is < 2000µs

Count > 3 → SCAN                    // invalid, burst is > 2000µs

Input == low → Count++, CHECK_START_LOW

Input == high → Count := 0, CHECK_START_HI
```

CHECK_START_HI: Make sure that the following *off* burst is also +/- 2000µs

```
Count < 3 && Input == low → SCAN    // invalid, burst is < 2000µs

Count > 3 → SCAN                    // invalid, burst is > 2000µs

Input == high → Count++, CHECK_START_HI

Input == low → Index := 0, READ
```

READ Start reading each bit by making sure it starts with 500µs *on* burst

Also check if 12+12 bits of data have been read, if so, signal main

```
Input == low → SCAN                      // invalid, burst is > 500µs

Input == high && Index == 24 → DataReady := 1, WAIT_FOR_MAIN

else always → Count := 0, READ_BIT
```

READ_BIT Read an individual bit, and determine whether it's zero or one

```
Count < 1 && Input == low → SCAN    // invalid, off-signal is < 1000µs

Count == 2 && Input == low → SCAN   // ambiguous, off-signal is 1500µs

Count > 3 → SCAN                    // invalid, off-signal is > 2000µs

Input == high → Count++, READ_BIT

Input == low && Count == 1 → Data[Index] := 0, Index := Index + 1, READ

else always → Data[Index] := 1, Index := Index + 1, READ
```

WAIT_FOR_MAIN Wait for main to acknowledge the new data and reset

```
DataReady == 0 → SCAN

else always → WAIT_FOR_MAIN
```