

INFRARED CAPTURE AND TRANSMISSION

GEDARE BLOOM

1. ABSTRACT

An effective means to perform capturing and replaying arbitrary infrared signals is presented. The architecture used to carry out the task is the Z16 ZNEO Contest Kit, an evaluation board by ZiLOG [1].

2. INTRODUCTION

Infrared is light of a wavelength that is not visible to the naked human eye. There are many sources of infrared light; everything that radiates heat also radiates infrared light [2]. With so much noise in the environment, an infrared receiver must know how to detect a signal. Infrared transmitters resolve the dilemma by pulsing the infrared light at a frequency that the receiver can distinguish from the noise. For infrared remote controls, the carrier's signal frequency is typically between 30 and 60 KHz [2]. When speaking of a remote control command, we will use *train* to mean the entire command, and *signal* to be the actual infrared pulses.

When speaking of serial communication, the time that a signal is pulsing is called a mark, and the time the transmitter is inactive is called a space [2]. To create binary digits from the signal that is pulsing at the carrier's frequency, the marks and spaces must be modulated. Typical encodings for infrared remote controls include bi-phase modulation (Manchester), pulse distance modulation, and pulse width modulation. Bi-phase modulation (Figure 1) allocates half the bit time to a mark and the remaining to a space; a bit is represented by a mark followed by a space, and its complement is a space followed by a mark. Pulse distance modulation (Figure 1) uses the same mark duration for each bit, but increases the space duration. Pulse width modulation (Figure 1) keeps space duration for each bit the same, but increases mark duration. The various modulation techniques are protocol-dependent, so the three encoding variants are found in different remote controls.

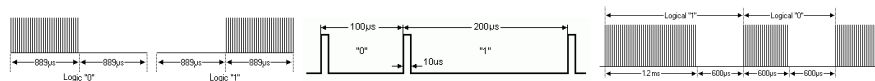


FIGURE 1. Bi-Phase, Pulse Distance, and Pulse Width modulation

In order to conserve power, the light signals that pulse during the mark phase is also pulse-width modulated with a period of $\frac{1}{frequency}$. The duty cycle of the pulse is typically between $\frac{1}{3}$ and $\frac{1}{4}$. Some encodings that use pulse distance modulation use only a single pulse to indicate a mark, but are prone to false positives [2].

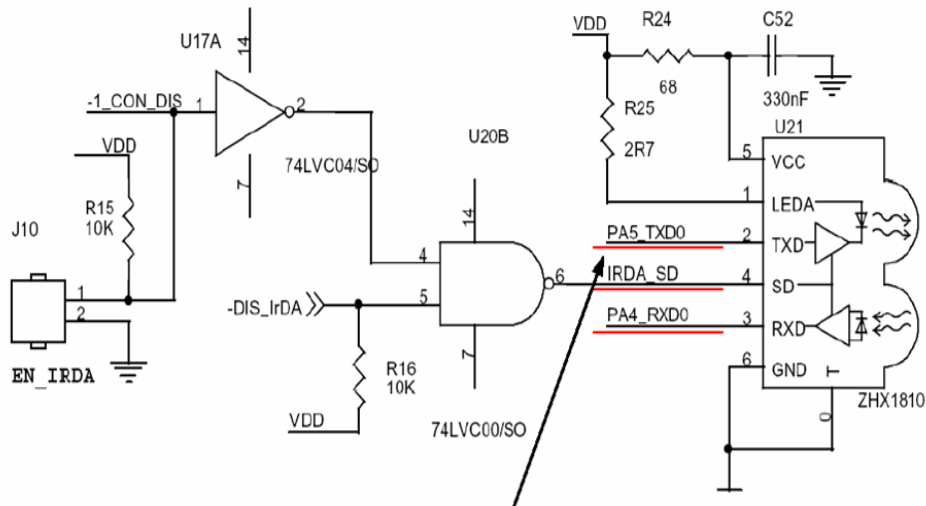
Date: April 17, 2007.

Serial Infrared (SIR), the IrDA standard [3] for lower frequency transmissions, uses Return-to-Zero Inverted (RZI) to define how to interpret the modulation. This means that the idle state is represented as a high voltage out of the receiver, and infrared pulses pull the voltage down. The transmitter is similarly high voltage when idle, and pulled down to transmit. The duration of the pull-down is the duty cycle of the pulse.

The remaining sections describe how to go about developing capture and re-transmission of infrared signals using the ZNeo Contest Kit and ZHX1810 SIR Transceiver.

3. READING INFRARED

The ZHX1810 SIR Transceiver is connected to the ZNeo and is expected to be used with the IrDA [3] protocol. However, examine Figure 2 and we see that the transceiver puts its signal out to PA5_TXD0 and PA4_RCV0, which are two of the general-purpose IO pins of the ZNeo.



What does “PA5_TXD0” mean?

FIGURE 2. ZNeo IR circuit schematic

By turning off the Alternate Function bits of Port A pins 4 and 5, the receive and transmit bits from the transceiver avoid going through the ZNeo’s UART. If the bits went through the UART, they would be formatted for serial communication; if the IR enable bit of the UART is set, then the signals will attempt to be interpreted as IrDA transmissions. In order to read and transmit raw signals, it is necessary to circumvent the UART. Also notice jumper J10 in figure 2; the jumper needs to be shunted to pull the input to U17A low. U17A is an inverter, so then both inputs

to U20B, a NAND gate, will be logic high. Thus, the output from U20B is logic low. SD is active high to shut down the transceiver, so life is good.

The next step is to decide what information is necessary in order to reproduce an infrared remote train. We need to know:

- the period of the modulated signal
- the duty cycle of the signal for each period
- the durations of marks and spaces

To determine the duty cycle, a gated timer¹ [4] is used. Gated timers count only when the timer input is active, so if the gated timer is configured for active low then it will count the duration of a signal. An interrupt can be generated at the inactive edge, in this case rising, and so the duty cycle can be captured each interrupt by reading the timer value. Averaging across successive signals will give a more stable value for the duty cycle.

To figure out the period and pulse durations, a capture-restart² timer³ [4] is used. Capture-Restart timers count until either the capture event occurs and the timer count is stored in the PWM field, or a reload timeout occurs. In either case, the timer is reset to 1 and counting continues. To utilize two timers, the input signal will need to be split to both timer input pins. The capture event should be the occurrence of a falling edge; the time between two consecutive falling edges of a mark phase is the period. The reload should occur when the mark phase is over, so the reload timeout should be at least as large as the longest expected period but not so large that a space is incorrectly missed. Period is the inverse of frequency, so find the smallest expected frequency, call it $frequency_{small}$. Then $reload_{value}$ should be:

$$(3.1) \quad Reload_{timeout} > \frac{1}{frequency_{small}}$$

$$(3.2) \quad Reload_{value} = round(Reload_{timeout} * sysclock)$$

$$(3.3) \quad = sysclock / frequency_{small} + 1$$

Add one at the end to adjust for the ‘greater than’ condition. For most commercial infrared remote controls, the smallest frequency is 30 KHz [2], so the reload value would be greater than the system clock divided by the frequency. Avoid making the reload value too large though, as mentioned previously, or spaces might be missed.

Figure 3 gives a generic state diagram for controlling timers in order to perform the receive operation. Algorithm 3.1 demonstrates how the capture-restart timer interrupt service routine should be configured. Now that the train is being read, a sample of the train can be retrieved and plotted. Using an oscilloscope, observe the actual signal that is received on pin PA4 and compare the two trains, see figures 4 and 6 for the plotted data and the actual train. The signals are quite close, considering the plotting method used is only accurate to about 0.5 milliseconds and each division is 5 milliseconds.

¹For MCUs with no gated timer, rising and falling edge-triggered interrupts may suffice

²if capture timer cannot reset timer on capture event, calculate differences between consecutive capture values, reload value will need adjusting

³For MCUs with no capture timer, edge interrupts can calculate period, durations will need special care

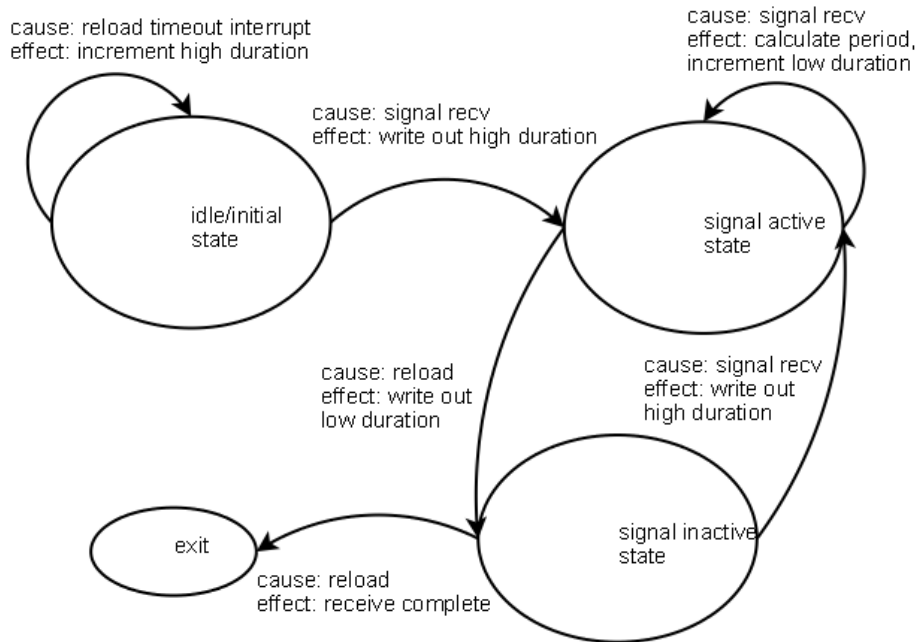


FIGURE 3. Receive timing states

4. TRANSMITTING INFRARED

In the previous section, an infrared train was successfully captured. So now begins the work of attempting to re-transmit the signal. Recall the information that was previously identified as necessary for transmitting a signal:

- period of the modulated signal
- duty cycle of the signal for each period
- durations of marks and spaces

The pulsing of a signal for a duty cycle across a period is easily implementing using a pulse-width modulated timer [4]. PWM timers output a signal during the pulse width phase, and then outputs nothing until the reload value is reached at which point counting resumes. So if the pulse width phase is loaded with the duty cycle value, and the reload value is loaded with the period, then all that remains is to attach the timer output to the transmit pin (PA5). The PWM timer should only be active during the mark phase of the signal. A space phase should deactivate the PWM timer. The durations of these phases vary across the train, so a one-shot timer [4] that reconfigures its reload value every timeout seems like a natural solution. Figure ?? shows a generic state diagram for the transmit operations.

However, there remains a slight problem; the solution needs 4 timers, and the ZNeo only has 3. Even if the ZNeo had 4, using all of them would leave no timers for other tasks which seems undesirable. Thankfully, the transmit and receive events can be kept mutually exclusive without losing functionality⁴. Now, with timers sharing duties between transmit and receive, it is necessary to also share interrupt

⁴It will no longer be possible to receive while transmitting

Algorithm 3.1: CAPTURERESTART_ISR(*void*)

comment: Algorithm for calculating period and transmit duration

comment: period and transmit duration in timer clock cycles

global unsigned int *high_cycles*, *low_cycles*

global unsigned int[] *low_duration*, unsigned int[] *high_duration*

global unsigned int *ldi*, *hdi*, *period*, *duty_cycle*

global unsigned char *prescalar*, *Status*

Status \leftarrow *BUSY* and not *RCV_COMPLETE*

if interrupt caused by Reload

{	if <i>high_cycles</i> = 0	{	comment: First high cycle detected										
	then		<table style="border-collapse: collapse;"> <tr> <td style="font-size: 2em; vertical-align: middle; padding-right: 5px;">{</td> <td style="padding-right: 5px;">if <i>low_cycles</i> > 0</td> <td style="padding-right: 5px;">then <i>low_duration</i>[<i>ldi</i>] \leftarrow <i>low_cycles</i></td> </tr> <tr> <td></td> <td></td> <td style="padding-right: 5px;">else <i>low_duration</i>[<i>ldi</i>] \leftarrow <i>duty_cycle</i></td> </tr> <tr> <td></td> <td></td> <td style="padding-right: 5px;"><i>low_cycles</i> \leftarrow 0</td> </tr> <tr> <td></td> <td></td> <td style="padding-right: 5px;"><i>ldi</i> \leftarrow <i>ldi</i> + 1</td> </tr> </table>	{	if <i>low_cycles</i> > 0	then <i>low_duration</i> [<i>ldi</i>] \leftarrow <i>low_cycles</i>			else <i>low_duration</i> [<i>ldi</i>] \leftarrow <i>duty_cycle</i>			<i>low_cycles</i> \leftarrow 0	
{	if <i>low_cycles</i> > 0	then <i>low_duration</i> [<i>ldi</i>] \leftarrow <i>low_cycles</i>											
		else <i>low_duration</i> [<i>ldi</i>] \leftarrow <i>duty_cycle</i>											
		<i>low_cycles</i> \leftarrow 0											
		<i>ldi</i> \leftarrow <i>ldi</i> + 1											
then	comment: Now check for idle state												

if *high_cycles* < *IDLE_THRESHOLD*

then *high_cycles* \leftarrow *high_cycles* + 1

else if *hdi* > 0

then $\left\{ \begin{array}{l} \textit{Status} \leftarrow \textit{not BUSY and RCV_COMPLETE} \\ \textit{high_cycles} \leftarrow 1 \\ \textit{low_cycles} \leftarrow 1 \end{array} \right.$

else *Status* \leftarrow not *BUSY* and not *RCV_COMPLETE*

{	if <i>high_cycles</i> = 0	{	comment: Not the first low cycle detected					
	then		<table style="border-collapse: collapse;"> <tr> <td style="padding-right: 10px;">comment: average period of modulation cycles for smoothness</td> </tr> <tr> <td style="padding-right: 10px;">if <i>period</i> = 0</td> <td style="padding-right: 10px;">then <i>period</i> \leftarrow <i>PWM_{data}</i></td> </tr> <tr> <td></td> <td style="padding-right: 10px;">else <i>period</i> \leftarrow $\frac{\textit{period} + \textit{PWM}_{\textit{data}}}{2}$</td> </tr> <tr> <td></td> <td style="padding-right: 10px;"><i>low_cycles</i> \leftarrow <i>low_cycles</i> + <i>PWM_{data}</i></td> </tr> </table>	comment: average period of modulation cycles for smoothness	if <i>period</i> = 0	then <i>period</i> \leftarrow <i>PWM_{data}</i>		else <i>period</i> \leftarrow $\frac{\textit{period} + \textit{PWM}_{\textit{data}}}{2}$
comment: average period of modulation cycles for smoothness								
if <i>period</i> = 0	then <i>period</i> \leftarrow <i>PWM_{data}</i>							
	else <i>period</i> \leftarrow $\frac{\textit{period} + \textit{PWM}_{\textit{data}}}{2}$							
	<i>low_cycles</i> \leftarrow <i>low_cycles</i> + <i>PWM_{data}</i>							
else	comment: First low cycle detected							

high_duration[*hdi*] \leftarrow *high_cycles* * *ReloadCycles* + *PWM_{data}*

hdi \leftarrow *hdi* + 1

high_cycles \leftarrow 0

service routines (ISRs). State variables could be used with an if-else structure in the ISR, but a more elegant solution presents itself with the use of function pointers. Now when the timers are configured for receive, two global function pointers

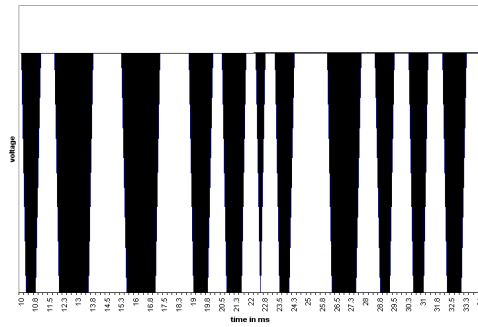


FIGURE 4. Data stored by capture algorithm

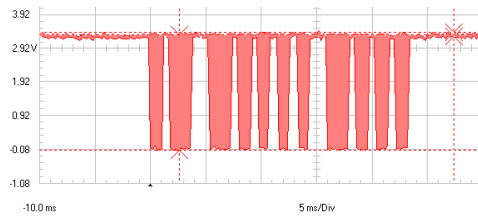


FIGURE 5. Oscilloscope data

will be updated to point to the functions that perform receive tasks. When timers are configured for transmit, the same global function pointers are updated to point to the functions that perform transmit tasks. Then each ISR will simply call the function pointer, for example:

```
// selector for various timer 0 ISRs.
void interrupt isr timer0( ) {
    SET_VECTOR(TIMER0,isr_timer0);
    DI();
    if(activeT0function)
        (*activeT0function)();
    else
        ; // fail silently?
    EI();
}
```

```
// selector for various timer 2 ISRs.
void interrupt isr timer2() {
    SET_VECTOR(TIMER2,isr_timer2);
    DI();
    if(activeT2function)
        (*activeT2function)();
    else
        ; // fail silently?
    EI();
}
```

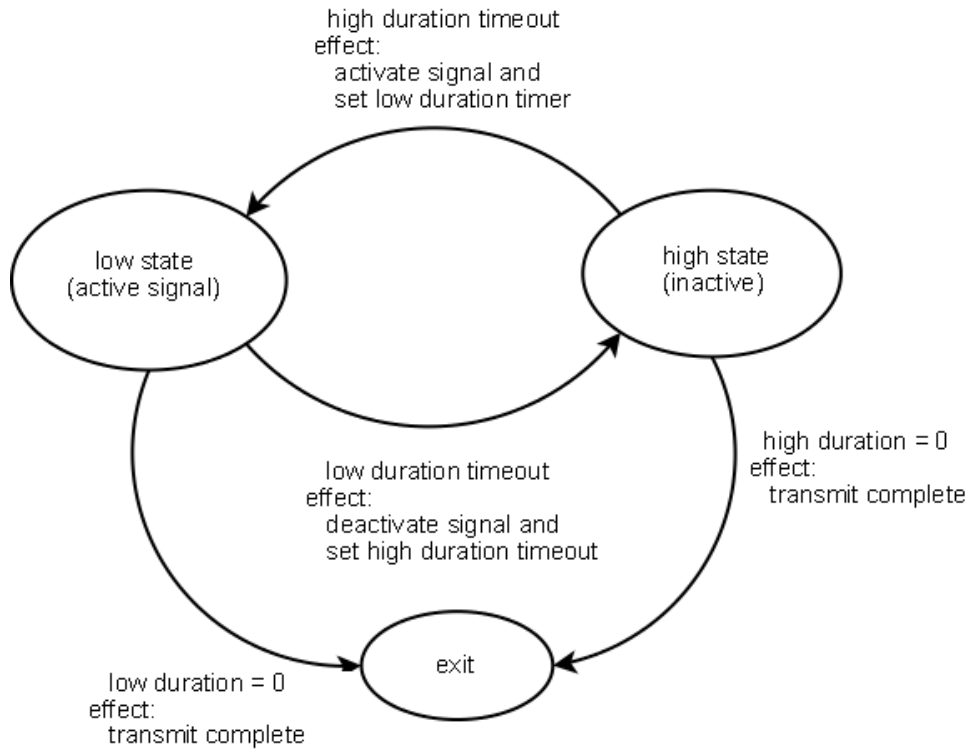


FIGURE 6. States of transmit

5. RESULTS

Figure 7 shows the comparison between the reconstructed train and the train captured at the receive pin. The trains look extremely similar, which is very promising. When tested, the reproduced signal does trigger the remote receiver, so success is achieved. However, notice that the reconstructed signal as seen on the transmit pin is the inverse of the signal on the receive pin, see Figure 8. This is not a problem, because the transceiver inverts the signal before transmitting. Transmission occurs when the transmit signal is pulled down.

REFERENCES

- [1] "Zilog web site," 2007. <http://www.zilog.com/>.
- [2] S. Bergmans, "Knowledge base: Ir remote control theory." <http://www.sbprojects.com>, 2001-2006. Oisterwijk, The Netherlands.
- [3] "Irda association," 2007. <http://www.irda.org/>.
- [4] "Zneo product specification ps0220.pdf," 2007. <http://www.zilog.com/software/zds2.asp#zneo>.

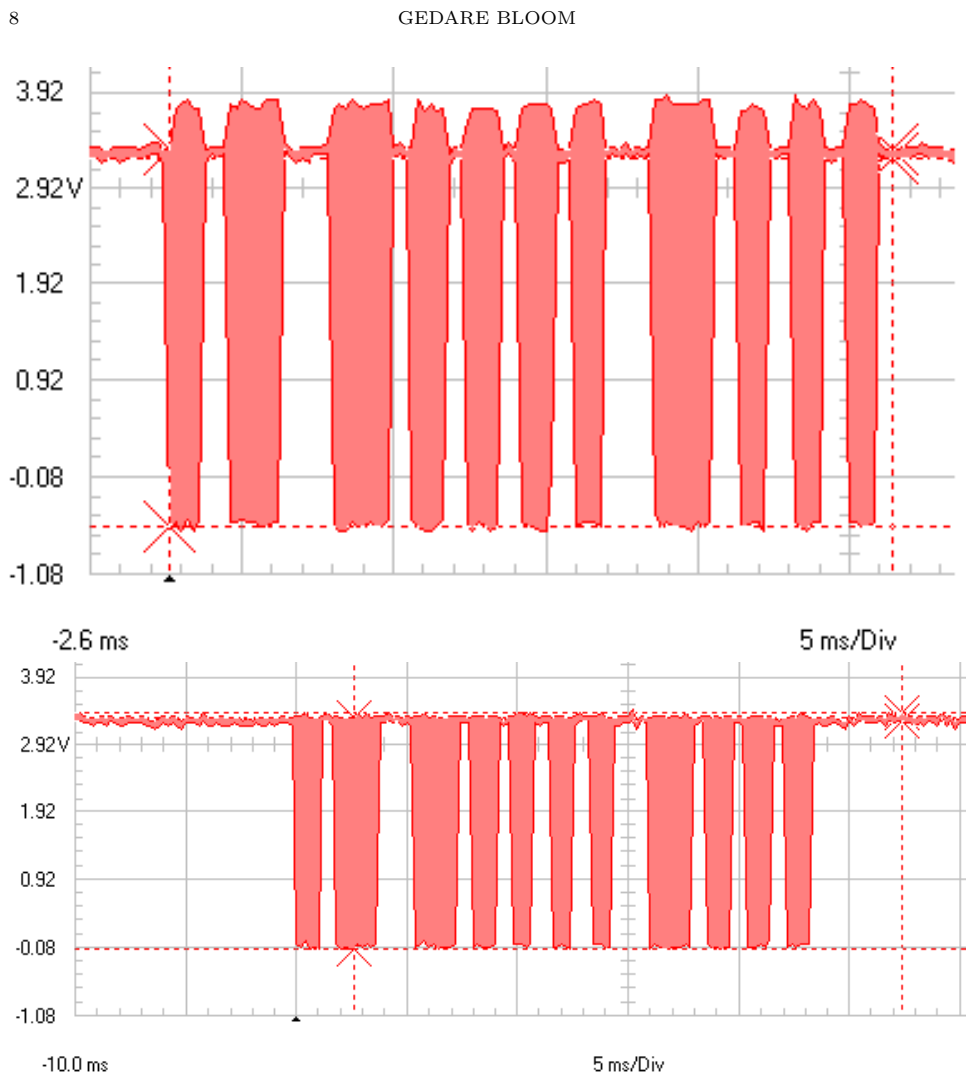


FIGURE 7. Reconstructed and Original trains

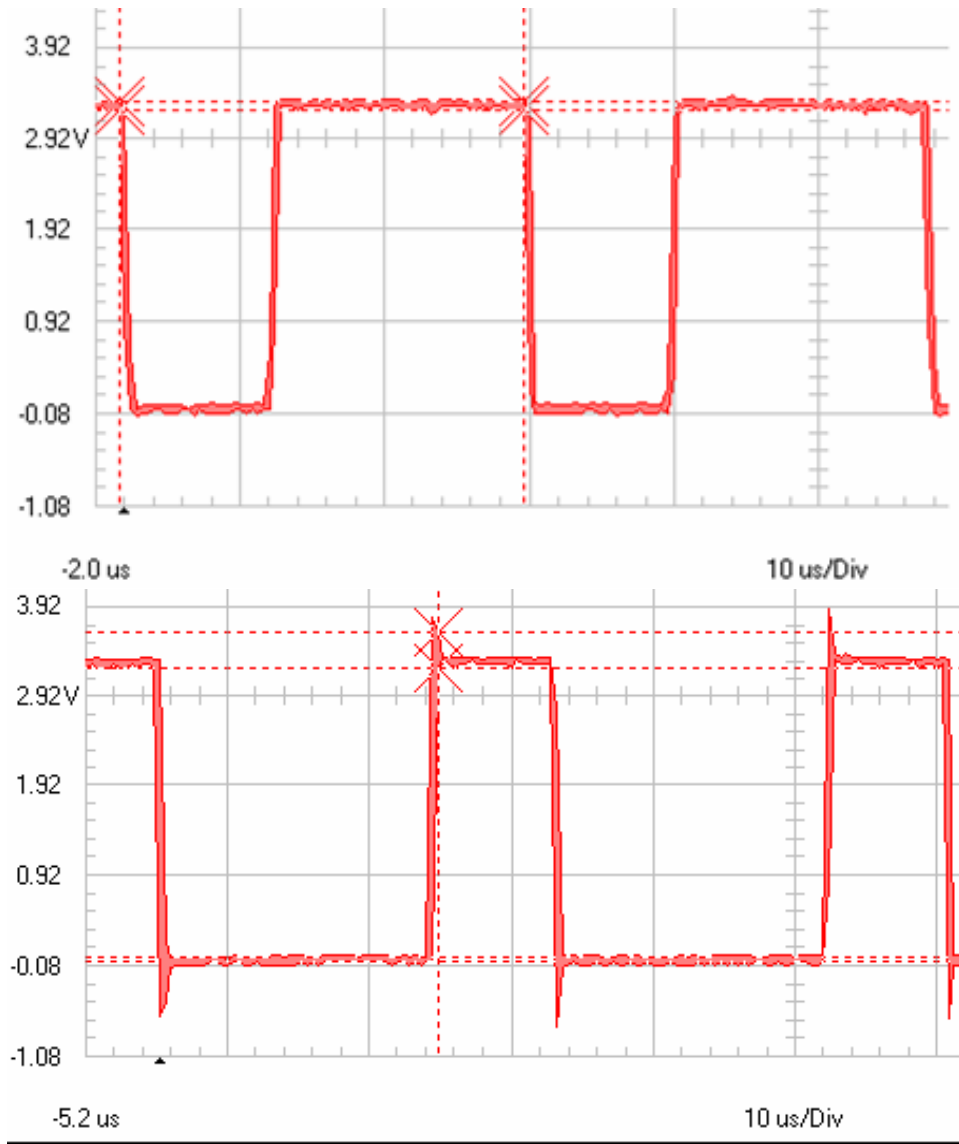


FIGURE 8. Receive and Transmit duty cycles differ