# ZBasic

Application Note

## AN-204 Input Capture and Multi-tasking for IR Remote Controls

### *Introduction*

An important feature of the ZBasic platform not found on many other microcontrollers is the ability to multi-task. This means that sensor input can be gathered in one task and then processed in another task. It is often the case that sensors don't always have data available at the time it is needed. Microcontrollers without multi-tasking, such as the Basic Stamp, need to poll the sensor to see if data is available. In a multi-tasking environment an interrupt or other input can be used to trigger a waiting task to retrieve the sensor data and then communicate the data to the control task when requested. In this application note, the sensor is an infrared (IR) receiver that is used to receive commands from an IR remote control.

### *Infrared Remote Control Fundamentals*

Remote controls send commands on a carrier frequency such as 38 KHz where the individual ones and zeros of the command are encoded as pulses using one of several standards such as RECS 80, RC5, and Sony SIRC. In this application note we are going to concentrate on the Sony SIRC protocol which uses pulse length modulation i.e. each bit to be transmitted is encoded as a burst of the carrier frequency followed by a gap, that is, the absence of carrier frequency. Each data packet begins with a start bit represented by a burst of 2.4ms duration. The start bit serves to identify the beginning of the data packet and to engage the automatic gain control (AGC) of the receiver. The data bits that follow are either a burst of 1.2ms or 0.6ms representing logic 1 and 0, respectively. The gap is always 0.6ms long.

There appears to be three versions of the Sony SIRC protocol that send data packets with either 12 bits, 15 bits or 20 bits of information. The most common is the 12-bit version which uses 5 bits for the device address and 7 bits for the device command with the LSB of the command transmitted after the start bit. Figure 1 depicts an example 12-bit data packet from a Sony remote where each block represents a burst of the 38 KHz carrier frequency. In this example packet Device Address 1 (TV) and Command 3 (Digit key 4) is transmitted. Transmissions are repeated every 45ms (measured from start bit to start bit) for as long as the key on the remote control is held down.

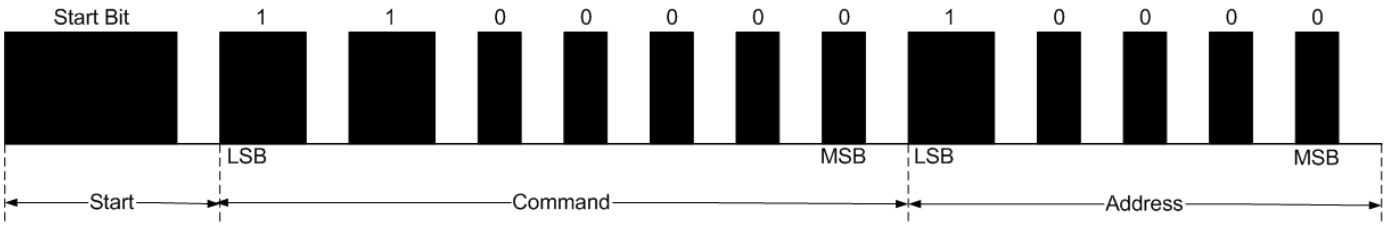


**Figure 1: Example Sony SIRC-12 Remote Control Data Packet**

An infrared receiver such as the Vishay TSOP34138 IR receiver (Mouser part #782-TSOP34138) demodulates the pulses from the carrier frequency and can present them to the microcontroller on a single input pin. Note that the output of this receiver is active low meaning that the bursts of the carrier frequency are output as a logic zero.

It is important that once the user presses a button on the remote control that the microcontroller is ready to receive and measure the pulse widths from the IR receiver. This can be achieved with the ZBasic chip using multi-tasking and the InputCapture system library subroutine. The InputCapture subroutine captures the measures and store the durations of low and high pulses from the IR receiver after being triggered by a falling leading edge. Because InputCapture suspends the current task until the packet of data has been gathered, it is important to use multi-tasking if you want the ZBasic microcontroller to perform other work while waiting for sensor input.

### *Hardware Schematic*

The EAGLE schematic shown in Figure 2 depicts the connection of a Vishay IR receiver (XR1) to the input capture port on a ZX-24 and ZX-40. Other required circuitry such as ZX-40 support hardware is omitted for clarity. Note that although it is a different pin on each chip, ZBasic automatically uses the correct input capture pin on the underlying



Published November 2005

AVR chip and you do not need to specify a pin number. The EAGLE library parts for the ZX-24 and ZX-40 can be downloaded from http://home.austin.rr.com/perks/micros/eagle/zbasic_lbr.zip.

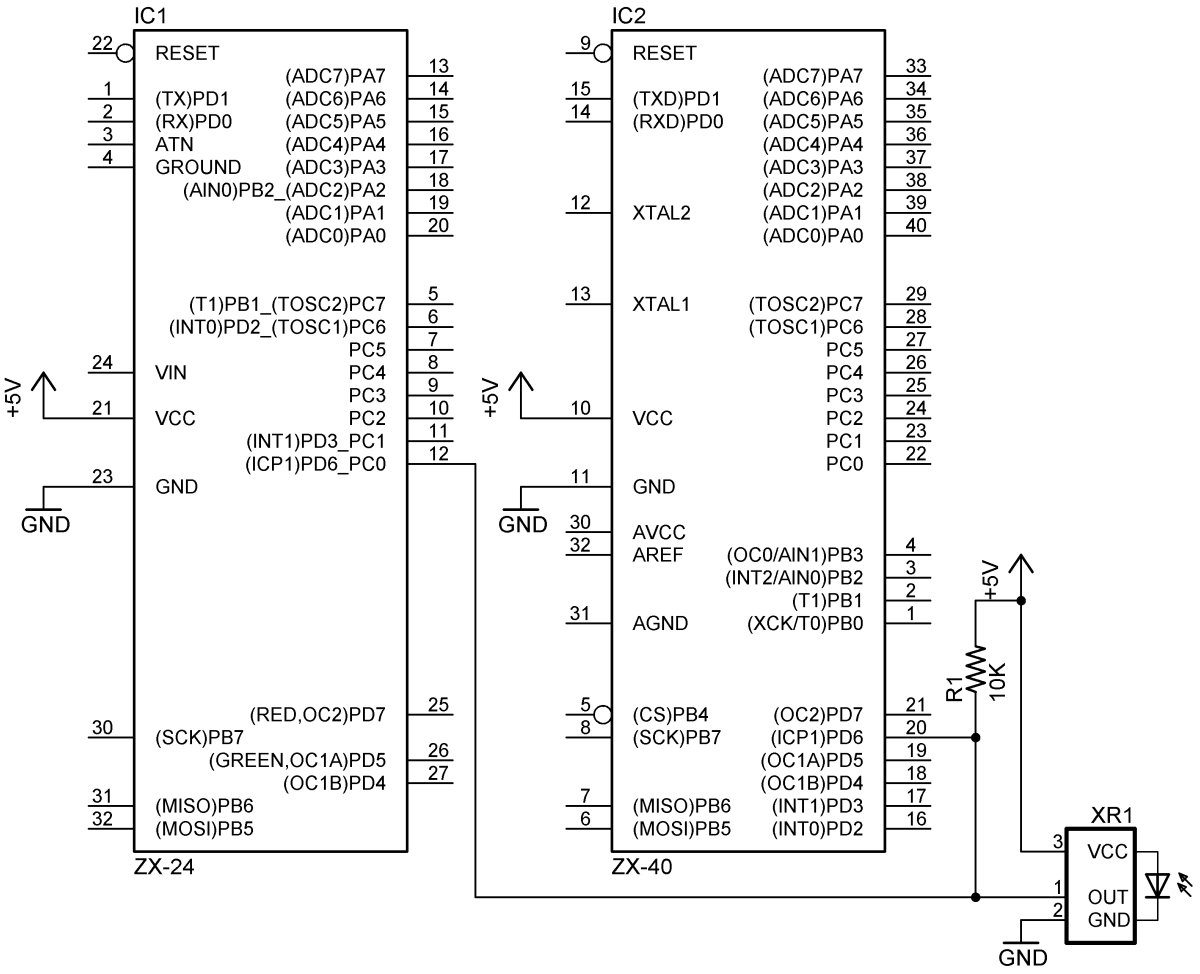


**Figure 2: Schematic for IR receiver connection to ZX-24 or ZX-40**

## *Software Interface*

The ZBasic example application (see AN-204.zip) consists of three ZBASIC source files and a ZBASIC project file. The IRTest.BAS file contains the main line code that retrieves IR commands and displays them on the PC using the ZBasic serial port monitor. The SonyIRCodes.BAS file contains a list of command and devices codes for Sony remotes. These codes may not be correct for all Sony remotes but it gives a good start. The IR.BAS file contains all of the functionality needed to retrieve infrared commands from a Sony remote.

The constants, variables, functions and subroutines in IR.BAS have been written in such a way that the module is reusable. There are three public routines for initializing, terminating and actual retrieval of an IR command code. The implementation, internal to the module, has three key parts: the call to the ZBasic InputCapture subroutine that retrieves the pulses from the IR receiver, a separate task that waits for the InputCapture subroutine to return with the array of pulses widths, and a queue that is used as the communication mechanism between the main control task and the IR task.

## *Example use of IR.BAS Module*

The code snippet below, taken from IRTest.BAS, shows how to retrieve IR commands. The details of the enclosing control code and command loop are not shown here for clarity. The call to GetIRCommand returns immediately with either an IR command byte or the constant value of NoCommand. All of the hard details of interfacing with the IR remote are hidden in InitRemote and GetIRCommand. The two subroutines InitRemote and ExitRemote are used to start and stop using the IR module. In most cases ExitRemote will never be called by a user application.



Published November 2005

```
      Dim command as Byte
      Call InitRemote()
      command = GetIRCommand()
      If command <> NoCommand Then
            Debug.Print CStr(command)
      End If
      Call ExitRemote()
```

## Initialization

The code below for InitRemote creates a queue to receive command codes and starts a new task to monitor the InputCapture port for IR remote command waveforms from the IR receiver. The queue size is a little larger than you would expect so that the queue can hold repeated commands and any missed commands while the control loop is doing other work.

```
'task stack for retrieving IR commands
Private Const IRStackSize as Byte = 60
Private IRStack(1 to IRStackSize) as Byte

' queue used for holding the command codes
Private Const queueSize as Integer = 15
Private IRQueue(1 to queueSize) as Byte

Public Sub InitRemote()
    ' open queue
    Call OpenQueue(IRQueue, QueueSize)
    ' start task
    CallTask "getIR", IRStack
End Sub
```

## Task to receive IR Commands

The code for the task to receive IR commands is shown below although some details have been removed for clarity. This task, implemented by the getIR subroutine, loops forever waiting for and retrieving infrared commands using the fetchCommand function. A valid command is added to the queue and then the task waits for the next pulse sequence from the IR receiver. A message is printed in the unlikely event that the queue is full and no more commands can be added to it.

```
Private Sub getIR()
    Dim command as Byte
   ' loop forever waiting for an IR command on the input capture port
    Do
        command = fetchCommand()
        ' put command on the queue if not full
        If command <> errorCommand Then
            If GetQueueCount(IRQueue) < GetQueueBufferSize(IRQueue) Then
                Call PutQueue(IRQueue, command, 1)
            Else
                Debug.Print "Lost IR command"
            End If
        End If
        Call Sleep(repeatTime)  ' remote repeats code after 45ms
    Loop
End Sub
```

## Using InputCapture

The fetchCommand function, shown below, is called by the IR task. FetchCommand calls InputCapture and waits for a stream of pulses from the IR receiver. It checks the pulse structure according to the Sony SIRC protocol and extracts the command code. The details for checking the SIRC protocol can be found in the attached source code and are not shown here for simplicity.



Published November 2005

```
' InputCapture is used to capture the pulse train into the array Pulses
Private Const pulsestreamLength as Integer = 25
Private pulses(1 to pulsestreamLength) as New UnsignedInteger

' CPU clock period in uS
Private Const clkPeriod_uS as Single = (1.0E+6 / 14.7456E+6)

' use original pulse widths as cheaper than converting to microsecond equivalents
Private Const startBitLength as UnsignedInteger= CUInt(2450.0 / clkPeriod_uS)'2450 us
Private Const zeroBitLength as UnsignedInteger = CUInt(625.0 / clkPeriod_uS) '625 us
Private Const oneBitLength as UnsignedInteger  = CUInt(1225.0 / clkPeriod_uS)'1225 us
Private Const gapLength as UnsignedInteger     = CUInt(575.0 / clkPeriod_uS) '575 us
Private Const deltaLength as UnsignedInteger   = CUInt(100.0 / clkPeriod_uS) '100 us

Private Function fetchCommand() as Byte
    Call InputCapture (pulses, pulseStreamLength, 0)

    ' create 7 bit command code, LSB first
    fetchCommand = 0
    Dim n as Integer
    For n = 1 to 7
        If withinRange(pulses(2*n+1), zeroBitLength, deltaLength) Then
            Call PutBit(fetchCommand, CByte(n-1), 0)
        ElseIf withinRange(pulses(2*n+1), oneBitLength, deltaLength) Then
            Call PutBit(fetchCommand, CByte(n-1), 1)
        End If
    Next
End Function
```

## *Getting the IR Command from the Queue*

The final part is the module public function that is used to return an IR command. This function, GetIRCommand, simply checks the queue and either returns the first item from the queue or a special command code if the queue is empty.

```
Public Const NoCommand as Byte = 255

Public Function GetIRCommand() as Byte
    GetIRCommand = NoCommand
    If GetQueueCount(IRQueue) > 0 Then
        Call GetQueue(IRQueue, GetIRCommand, 1)
    End If
End Function
```

## *Author*

Mike Perks is a professional software engineer who became interested in microcontrollers a few years ago. Mike has written a number of articles, projects and application notes related to ZBasic, BasicX and AVR microcontrollers. Mike is also the owner of Oak Micros which specializes in AVR-based devices including his own ZX-based products. You may contact Mike at mikep@oakmicros.com or visit his website http://oakmicros.com.