



Column #146, November 2007 by Jon Williams:

## Dimming the Lights Fantastic

*Wow... it is, once again, that time of year; the time of year where the air is crisp, the colors are vibrant, and holiday lights decorate neighborhoods across the country. It seems like – in my suburban Los Angeles neighborhood, anyway – that November 1st has become the official start of the Christmas decorating season. Well, that's okay by me, especially after bumping into Vixen a few months ago. You may remember that back in May we used Vixen as an animatronics controller. This month we're going to build an eight-channel lighting dimmer that can be controlled with Vixen, or a BASIC Stamp, remote SX, or even a Propeller processor if we like.*

Lamp dimming – specifically 120 VAC lamp dimming – had been on my mind for quite a long time, and this year I finally jumped in and tackled the process. It started with a four-channel device called the FC-4 that I designed for EFX-TEK. My friends who are Christmas enthusiasts asked that I create a similar device that could be expanded to 64, 96, or even 128 channels; that's what this month's project is all about. The controller has eight dimming channels, and is addressable so that 16 boards can happily co-exist on the same link; this gives us 128 dimmable channels; probably enough for most home lighting displays.

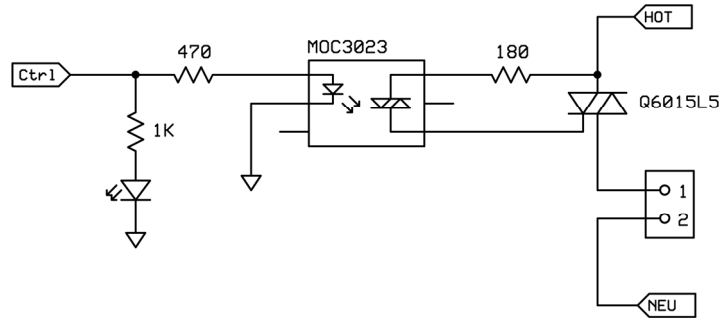
### AC Control

As this project is a bit involved let's just jump right in. We'll start with switching AC using a digital circuit. Many of us have used mechanical relays to switch AC, but they're noisy and can arc, ultimately forcing us to replace the relay due to contact pitting (or worse, contact fusing) – and mechanical relays have only two states: on and off. The advent of the solid state relay is a boon to those of us that want to switch AC with digital circuits. Crydom is a

## Column #146: Dimming the Lights Fantastic

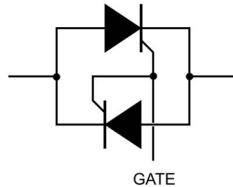
popular manufacturer of solid state relays providing devices that can take a direct TTL input signal and switch voltages from 12 to 230 VAC at several tens of amperes.

For Christmas tree lights we don't need to switch that much current; this means we can create our own solid state relay circuit. Figure 146.1 illustrates a very common circuit used to switch AC with a TTL control signal. Note that there is optical isolation between the control and output sides – this is very important for safety.



**Figure 146.1: SSR Schematic**

A TTL high on the control pin will cause the LED in the MOC3023 to light which in turn causes the triac side of this device to conduct. This allows the output power triac to be gated and conduct – the triac acts like a switch in the AC line between hot and neutral. When the control signal is removed and the AC voltage hits the zero-cross point the output triac will shut off.



**Figure 146.2: Triac Construction**

That last point is very important to understand, especially when it comes to lamp dimming. A triac is a solid state switch that is designed for AC circuits. It's convenient to think of a triac like two head-to-toe SCRs connected together with a common gate, as in Figure 146.2. When the gate signal is applied one side or the other will conduct (two sides are required for AC). When the gate is removed and the voltage passes through zero the triac will shut down. This

SCR-like (self-latching) behavior is the reason you can't switch DC with a triac – well, more accurately, you can switch DC on but you can't turn it off without removing power.

### AC Lamp Dimming

If a triac stays on until the gate is removed and voltage passes through zero, how do we use it as an element in a dimmer circuit? What we have to do is monitor the AC voltage for the zero-cross and then gate the triac sometime after, and within the same half cycle. If we gate the triac about half-way through one half of the AC cycle a lamp connected to this output will be at about 50% brightness.

Figure 146.3 illustrates an AC waveform showing the zero-cross points; there are two per cycle, so our dimmer control circuit will actually fire the triac at a 120 Hz rate. The blue area under the curve indicates the time when the triac is conducting. Note the hold-off period; the shorter this period, the brighter the lamp will light.

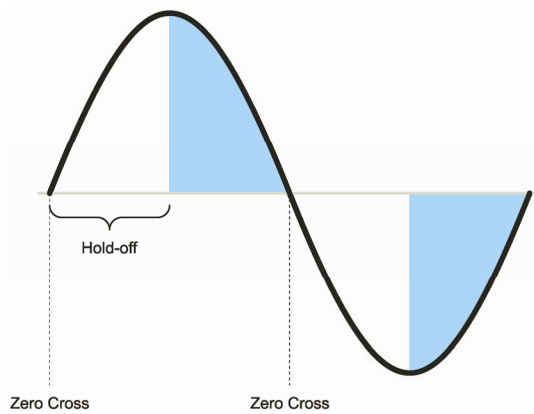
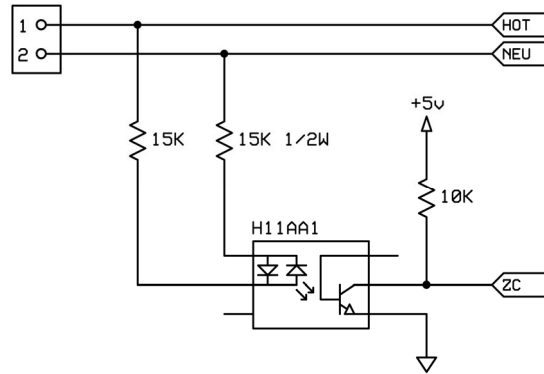


Figure 146.3: AC Switching

As you might expect, we'll use another opto-isolator to monitor the power line for zero-cross. Figure 146.4 shows a circuit that I've seen used in many home-brew lamp dimming circuits and it works quite well. The output on the ZC line has a high-going pulse every 8.333 milliseconds (120 Hz) that occurs very near the zero cross (it actually straddles the zero-cross point as the LEDs in the H11AA1 have a 1.2v forward voltage). Our program will use this pulse to start the hold-off timer for each dimming channel.

## Column #146: Dimming the Lights Fantastic



**Figure 146.4: Zero-Cross Detector**

In order to control eight dimmers and get their values from an external device via serial input we will construct an ISR (interrupt service routine) that handles the receive UART and the dimmers. I selected 38.4 kBaud for the input as this lets us refresh 128 channels in under 50 milliseconds (good for zippy displays), and the math works out pretty cleanly: with a 26.042 uS bit time we need to run the interrupt every 6.51 uS in order to do 4x sampling of serial bits. Fortunately for us, 6.51 will divide cleanly into 32.55 uS which is 1/256th of each 60 Hz half cycle. This allows us to set channel brightness with a byte, and as it takes a full byte we can do a little code trickery to construct the hold-off period.

Within the ISR we'll have a divider that runs the dimmer processing every fifth cycle. It looks like this:

```
Check Dimmer Tix:
  ASM
  BANK  dimmerTix
  INC   dimmerTix
  CJB   dimmerTix, #5, Dimmer_Done
  CLR   dimmerTix
```

This is pretty easy – we increment dimmerTix and when it hits five we will process the dimmers (resetting dimmerTix before we do). If you haven't jumped in to SX assembly let me encourage you to give it a try. Honestly, I'm a poster boy for the purpose of SX/B: to help BASIC Stamp users migrate from BASIC only to mixed and assembly-only projects. Remember that SX/B compiles to straight assembly, so you can always write something in BASIC and look at the compiled output to see how the translation is made. Using this very

process I have been adding a lot of assembly segments to my programs where absolute efficiency is key; you can, too.

Okay, now that it's time to process the dimmers what we need to do is check to see if we're at the zero cross point. This is easy: the ZC input pin will be high if we are. Let's assume that's the case and that we're at the beginning of a new half-cycle.

```
Dimmer Service:
  BANK  dimmer
  JNB   ZCross, Update_Triacs
```

```
Zero Cross:
  CLR  Lamps
  MOV  acc1, chan1
  MOV  acc2, chan2
```

At the zero-cross point the program clears all of the triac gate control pins (on port RC). Then the current brightness level for each channel is reloaded into an accumulator for that channel. Note that I'm only showing two channels above, but the program actually has eight.

Okay, now for the fun stuff. The program uses a PWM technique that is very clever and dirt simple to implement.

```
Update Triacs:
  INC  acc1
  SNZ
  SETB Lamp1
  INC  acc2
  SNZ
  SETB Lamp2
```

Each accumulator is incremented and if it rolls over to zero the corresponding triac gate is enabled. So you see, the higher the channel value, the sooner it will roll over to zero and allow the triac to be gated. And as we saw above, the earlier we gate the triac relative to the zero-cross the brighter the lamp will be.

Pretty neat, huh? Yeah, I think so, too.

### Interrupt Fine Tuning

Being a math wizard you've probably figured out that with 4x sampling of 38.4K serial bits we should run the ISR at a rate of 153,600 times per second. You're right. Why, then, does the program use the oddball value of 153,826?

## Column #146: Dimming the Lights Fantastic

Here's the deal... we're running the SX at 50 MHz which means that each instruction takes just 20 nanoseconds. If we divide 6.51 microseconds by 20 nanoseconds we get 325.5 – whoops, this value is bigger than a byte so it won't fit into the RTCC register which is what controls periodic interrupts. Remember that the SX/B compiler is very smart, so what it does is set the RTCC prescaler to 1:2; this has the effect of clocking the RTCC every 40 nanoseconds (25 MHz); now the rollover value is 162.7 (which gets rounded to 163).

No problem, right? Well, no, but then what happens is that the dividing up of the AC half-cycle is just a tiny bit out of whack. It doesn't hurt anything, but what we'll see is a very dim control LED when the channel is in fact supposed to be off; this bothered me enough to fix it.

Here's what I did: I took the RTCC clock of 40 nanoseconds and multiplied it by 163 (cycles for each interrupt) to get 6.52 uS. Into this value I divided 6.51 uS (ideal timing) to get 1.001472 and multiplied that by the initial ISR setting of 153,600 – the final result is 153,826. This eliminates the ghosting on the control output LEDs (off is now off) and does not impact the receive UART as the change is only about 0.14 percent.

The important lesson is that “ideals” are sometimes less than idea in practical application, and that we shouldn't be afraid of tweaking to get best performance from a piece of code.

### External Control

This board is designed to be a slave of another device; that device could be a BASIC Stamp on a Board of Education or the PC program Vixen using an RS-485 multi-drop link. The reason for the RS-485 link is to allow significant distance between the PC and the actual dimmer board. We could, for example, use a PC in the back of the house to control one or more of these boards installed in the family room near the Christmas tree.

Figure 146.5 shows the associated serial connections for the board. For local control (LTC485 chip removed) with a BASIC Stamp we can use X2 and X3 (for daisy-chaining) and Open-True mode communications. For long distance control we can use RS-485 (LTC485 installed) serial over standard CAT5 networking cable. I'd love to convince you that I'm very smart and came up with this idea, but I didn't; I “liberated” this idea from my good friend, Peter (who also maintains the SX-Key IDE for Parallax), after seeing it in use in his modular animatronics control system (see [www.socalhalloween.com](http://www.socalhalloween.com)). CAT5 cable works well because it uses twisted pairs and is designed for much higher data rates. In Peter's system he actually carries DC power (12 volts) to his boards through the cable as well.

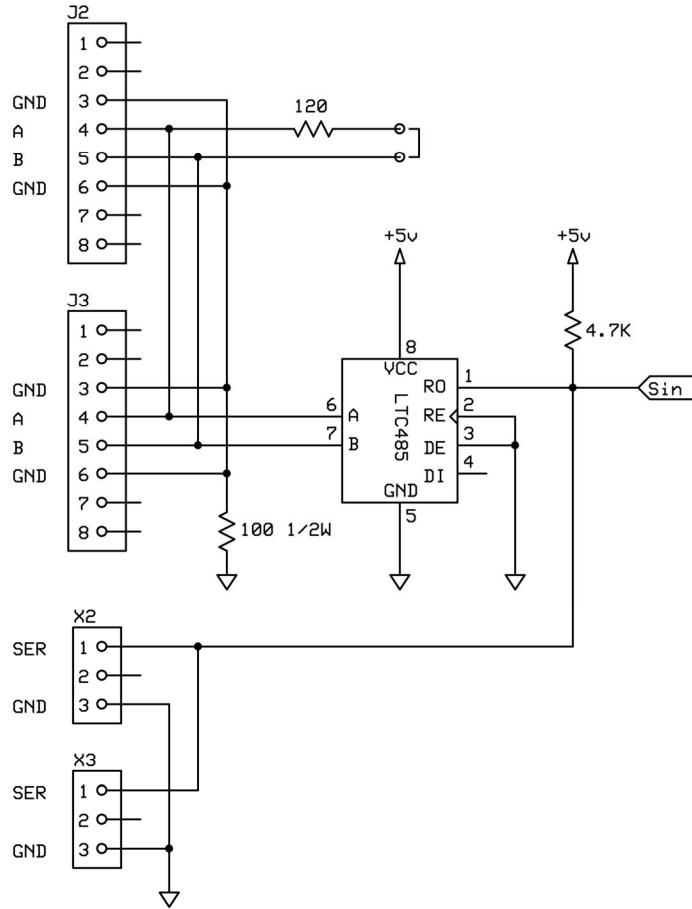


Figure 146.5: Serial Connections

The RS-485 interface is standard, and is set up for receive only. A jumper on the board allows for a 120-ohm terminating resistor as this is required on the final node in an RS-485 chain. Note that the 100-ohm resistor to ground is optional, and only needed if you decide to have common ground between all boards. For more details on this I recommend Jan Axelson's excellent book, *Serial Port Complete*.

## Column #146: Dimming the Lights Fantastic

### Decoding the Serial Stream

Like the animatronics controller from May, this project uses a break between streams to synchronize the slaves to the data within the stream. The process is simple: after a break period of at least one millisecond the board expects to see up to 16 packets of eight-byte lamp dimmer values. The address of the board determines which group of bytes are plucked from the stream and transferred into the dimmer control registers (chan1 – chan8).

The break period is detected by a small section of code in the ISR. Each time through it will increment a counter (breakTmr) when the receive line is idle, otherwise it will clear this counter and the associated flag (hasBreak). When the value of breakTmr reaches 154 (about a millisecond), the hasBreak flag is set and the timer is restarted.

```
Check_Break_Timer:
  ASM
  BANK 0
  JNB RX, RX Has Bit
  INC breakTmr
  CJB breakTmr, #BreakCnt, Check_Break_Exit
  SETB hasBreak
  CLR breakTmr
  JMP Check Break Exit

RX Has Bit:
  CLRB hasBreak
  CLR breakTmr

Check Break Exit:
  ENDASM
```

Remember that a start bit and zero bit in this system will pull the RX line low, hence the use of JNB (jump if this bit is 0). By installing this code in the ISR the foreground program can simply look for the setting of the hasBreak flag to indicate that a valid break period has been detected.

```
Main:
  breakTmr = 0
  hasBreak = No
  DO WHILE hasBreak = No
  LOOP
```

Once the break is detected we need to flush the receive UART before proceeding. This seemingly innocuous step caused me three days of headaches trying to track down a “fluttering” problem with my lamps. While I had been clearing the rxReady flag I neglected to clear rxCount – this value controls bits coming into the [free-wheeling] UART. What was happening, I believe, is that garbage values in rxCount caused misaligned data and corrupted

## The Nuts & Volts of BASIC Stamps 2007

my dimmer values. It was horribly frustrating – I even called Peter to ask him to review my code to see if I was missing something. Sometimes just talking through a problem with a friend can be helpful and I finally determined the problem while explaining to Peter how the program works.

```
Flush Uart:
  rxCount = 0
  rxReady = No
```

The next step is to read the board address and multiply that by eight so that we know how many bytes in the stream to skip.

```
Align Packet:
  idx = 0
  idx.0 = ~Addr.3
  idx.1 = ~Addr.2
  idx.2 = ~Addr.1
  idx.3 = ~Addr.0
  idx = idx << 3
```

This looks like more work than what's needed; I did it this way to “fix” a board layout issue – what I wanted to do is have the LSB of the switch be to the right as I could read the markings on the SX28. As the LSB switch is actually connected to RA.3 instead of RA.0 (see Figure 146.6), we manually construct the value to mirror the bits of the address. There may be some chunk of cute code out there to mirror a byte, but this bit-by-bit approach seemed to make the most sense to me. The board address, now in `idx`, is multiplied by eight (`<< 3`) to determine the number of bytes to skip. And here's how we do that:

```
Skip Bytes:
  DO WHILE idx > 0
    tmpB1 = RX_BYTE
    IF hasBreak = Yes THEN Main
    DEC idx
  LOOP
```

As you can see, this is all very straightforward – as long as `idx` is greater than zero we'll pull a byte from the receive buffer and just toss it. After bytes start coming in we can check the `hasBreak` flag again; this will cause the program to jump back to the beginning if the stream gets interrupted.

Column #146: Dimming the Lights Fantastic

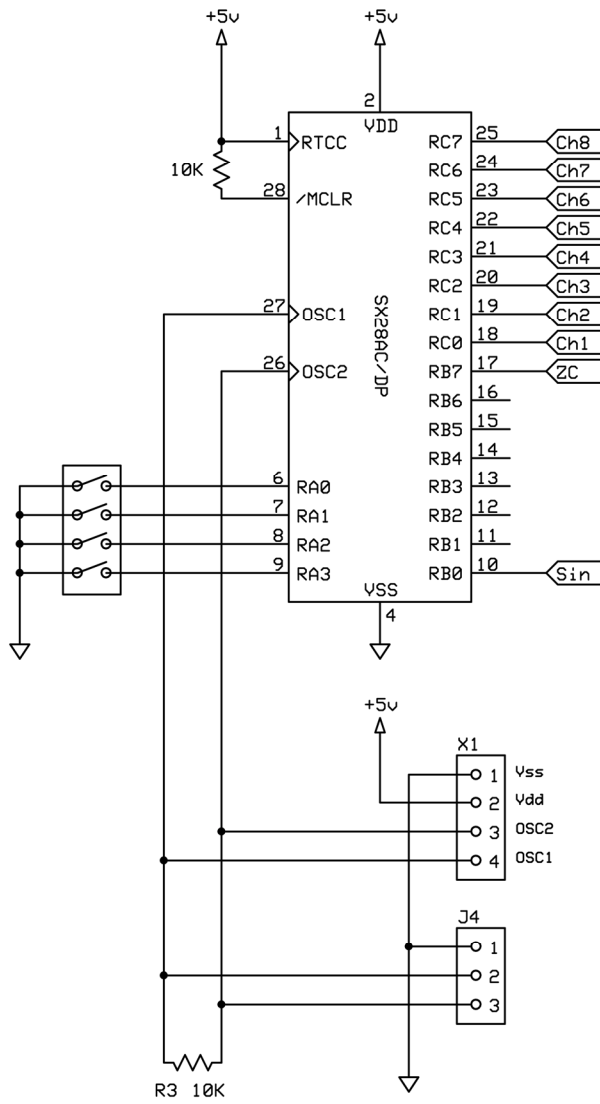


Figure 146.6

The final step is to pull our eight bytes into the dimmer control registers. The rest automatic as the dimmers are already running in the ISR.

```
Get_Levels:
  FOR idx = 0 TO 7
    IF hasBreak = Yes THEN Main
    dimmer(idx) = RX BYTE
  NEXT
  GOTO Main
```

### Construction

It would take a very hearty soul to tackle this project using point-to-point wiring – and with 120 VAC involved let me suggest that you don't. As with other projects I used ExpressSCH and ExpressPCB to create the PCB for the dimmer. Those of you with a lot of layout experience may find the board a little generous (i.e., it could be smaller); I decided to err on the side of caution considering what's involved.

If you're new to PCB layout and decide to give ExpressPCB a run, let me VERY STRONGLY suggest that you start with ExpressSCH, check it for netlist errors, and then create the PCB by linking to the schematic. What I like about later versions of ExpressPCB is the ability to do flooded planes. The only problem when using a plane with a prototype board (one without solder mask) is that the small spacing between traces and the plane is a magnet for solder bridges. I expand the pad and trace spacing to 0.015" (this done in Layout\Board Properties menu), and use a very clean, fine-tipped soldering iron when I work. I also proceed very slowly, checking connections near the plane through a loupe to make sure that I don't have any bridges.

Let me make very important point about this project: it involves 120 VAC which can be dangerous if mistreated. Also, prototype PCBs cannot handle large currents, so you should put an inline fuse (5A) on the AC power input (TB1) and ensure that your Christmas light strings are likewise fused. Finally, the whole works should be assembled in a fireproof enclosure using suitable stand-offs from the panel. The holiday season is supposed to be joyous – let's keep it that way.

### Happy Holidays

Well, that wraps up this one – order your boards and parts and start building. Whether you use it with a BASIC Stamp or a big control application like Vixen, I'm sure you'll find this project a lot of fun.

Happy Holidays, and until next time, Happy Stamping!

**Column #146: Dimming the Lights Fantastic****Parts List**

<b>Light Controller Bill of Materials</b>		
<b>Designator</b>	<b>Value</b>	<b>Source</b>
C1, C2	47	Mouser 647-UVR1V470MDD
C3	0.1	Mouser 80-C315C104M5U
D1-D9	LED	Mouser 859-LTL-4222N
J1	2.1 mm barrel	Mouser 806-KLDX-0202-A
J2, J3	RJ-45	Mouser 571-5202514
J4	0.1 strip socket	Mouser 506-510-AG91D
JP1	0.1 strip header	Mouser 517-6111TG
Jumper	0.1 jumper	Mouser 535-ME-100
PCB		ExpressPCB.com
Q1-Q8	Q6015L5	Mouser 576-Q6015L5
R1, R12, R15, R18, R21, R24, R27, R30, R33	1K	Mouser 299-1K-RC
R2, R3, R9	10K	Mouser 299-10K-RC
R4	120 1/4 W	Mouser 291-120-RC
R5	100 1/2 W	Mouser 293-100-RC
R6	4.7K	Mouser 299-4.7K-RC
R7, R8	15K 1/2 W	Mouser 293-15K-RC
R10, R13, R16, R19, R22, R25, R28, R31	470 1/4 W	Mouser 299-470-RC
R11, R14, R17, R20, R23, R26, R29, R32	180 1/4 W	Mouser 291-180-RC
S1	28-pin DIP	Mouser 571-1-390261-9
S2	8-pin DIP	Mouser 571-1-390261-2
S3-11	6-pin DIP	Mouser 571-1-390261-1
SW1	SPDT slide	Mouser 506-SSA12
SW2	4-pos DIP	Mouser 106-1204-EV
TB1-TB9	Terminal Block	Mouser 571-2828362
U1	SX28AC/DP	Parallax SX28AC/DP
U2	LTC485	Jameco 299807
U3	H11AA1	Mouser 782-H11AA1
U4-U11	MOC3023	Mouser 859-MOC3023
VR1	LF50CP	Mouser 511-LF50CP
X1	0.1 R/A Male Header	Mouser 517-5111TG
X2,X3	0.1 Male Header	Mouser 517-6111TG

## Source Code

```

' =====
'
' File..... Lamp Dimmer 8x.SXB
' Purpose...
' Author.... Jon Williams, EFX-TEK
'           Copyright (c) 2007 Jon Williams
'           Some Rights Reserved
'           -- see http://creativecommons.org/licenses/by/3.0/
'           -- portions inspired by works of Phil Short
'           -- ISR UART code originally by Chip Gracey
' E-mail.... jwilliams@efx-tek.com
' Started...
' Updated... 23 SEP 2007
'
' =====
'
' -----
' Program Description
' -----
'
' -----
' Conditional Compilation Symbols
' -----
'
' -----
' Device Settings
' -----
'
DEVICE          SX28, OSCHS3, TURBO, STACKX, OPTIONX, BOR42
FREQ            50 000 000
ID              "Lamps_8x"
'
' -----
' IO Pins
' -----
'
Addr           PIN    RA    INPUT  PULLUP  ' board address, 0 - 15
RX             PIN    RB.0 INPUT  NOPULLUP
ZCross        PIN    RB.7 INPUT  NOPULLUP SCHMITT
UnusedRB1     PIN    RB.1 INPUT  PULLUP
UnusedRB2     PIN    RB.2 INPUT  PULLUP
UnusedRB3     PIN    RB.3 INPUT  PULLUP
UnusedRB4     PIN    RB.4 INPUT  PULLUP

```

## Column #146: Dimming the Lights Fantastic

```
UnusedRB5    PIN    RB.5  INPUT  PULLUP
UnusedRB6    PIN    RB.6  INPUT  PULLUP

Lamps        PIN    RC    OUTPUT
Lamp1        PIN    RC.0  OUTPUT
Lamp2        PIN    RC.1  OUTPUT
Lamp3        PIN    RC.2  OUTPUT
Lamp4        PIN    RC.3  OUTPUT
Lamp5        PIN    RC.4  OUTPUT
Lamp6        PIN    RC.5  OUTPUT
Lamp7        PIN    RC.6  OUTPUT
Lamp8        PIN    RC.7  OUTPUT

' -----
' Constants
' -----

Yes          CON    1
No           CON    0

IsOn         CON    1
IsOff        CON    0

Idle         CON    1           ' RX line state
HasBit       CON    0

' Dividers for ISR UART
' -- values set for 6.51 uS interrupt rate

Baud9600     CON    16
Baud19K2     CON    8
Baud38K4     CON    4

Baud1x0      CON    Baud38K4
Baud1x5      CON    Baud1x0 * 3 / 2

BreakCnt     CON    154           ' ~ 1 ms

' -----
' Variables
' -----

flags        VAR    Byte
  isrTic     VAR    flags.0           ' isr marker
  hasBreak   VAR    flags.1           ' break space detected
  rxReady    VAR    flags.2           ' byte is waiting

breakTmr     VAR    Byte           ' (ISR) counter for sync
```

## The Nuts & Volts of BASIC Stamps 2007

## Column #146: Dimming the Lights Fantastic

```

dimmerTix    VAR    Byte    ' (ISR) divider for dimming
idx          VAR    Byte    ' loop controller

tmpB1       VAR    Byte    ' sub/func work var
tmpW1       VAR    Word

rxSerial    VAR    Byte (3)  ' rx serial data
rxCount     VAR    rxSerial(0) ' rx bit count
rxDivide    VAR    rxSerial(1) ' bit divisor timer
rxBuf       VAR    rxSerial(2) ' one byte buffer

dimmer      VAR    Byte (16)  ' bank for dimmer vars
chan1       VAR    dimmer(0)  ' channel brightness
chan2       VAR    dimmer(1)
chan3       VAR    dimmer(2)
chan4       VAR    dimmer(3)
chan5       VAR    dimmer(4)
chan6       VAR    dimmer(5)
chan7       VAR    dimmer(6)
chan8       VAR    dimmer(7)
acc1        VAR    dimmer(8)  ' brightness accumulator
acc2        VAR    dimmer(9)
acc3        VAR    dimmer(10)
acc4        VAR    dimmer(11)
acc5        VAR    dimmer(12)
acc6        VAR    dimmer(13)
acc7        VAR    dimmer(14)
acc8        VAR    dimmer(15)

' =====
INTERRUPT NOCODE 153 826          ' every 6.51 uS
' =====

Mark ISR:
ASM
    BANK 0                        ' (1)
    SETB isrTic                   ' (1)
ENDASM

' -----
' RX packet break monitor
' -----

Check Break Timer:
ASM
    BANK 0                        ' (1)
    JNB RX, RX_Has_Bit            ' (2/4) check RX line
    INC breakTmr                  ' (1) idle, inc timer
    CJB breakTmr, #BreakCnt, Check_Break_Exit ' (4/6) = BreakCnt?

```

## Column #146: Dimming the Lights Fantastic

```
    SETB  hasBreak          ' (1)  yes, mark it
    CLR   breakTmr         ' (1)  and restart timer
    JMP   Check Break Exit  ' (3)

RX Has Bit:
    CLRB  hasBreak        ' (1)
    CLR   breakTmr       ' (1)

Check Break Exit:
    ENDASM

' -----
' RX UART
' -----

Receive:
    ASM
    BANK  flags          ' (1)
    JB   rxReady, RX Done ' (2/4) skip if byte waiting
    BANK rxSerial       ' (1)
    MOVB C, RX          ' (4)   sample serial
input
    TEST  rxCount        ' (1)  receiving now?
    JNZ  RX_Bit         ' (2/4) yes, get next bit
    MOV  W, #9          ' (1)  no, prep for next
byte
    SC   ' (1/2)

    MOV  rxCount, W     ' (1)  if start, load bit count
    MOV  rxDivide, #Baud1x5 ' (2)  prep for 1.5 bit periods

RX Bit:
    DJNZ  rxDivide, RX Done ' (2/4) complete bit cycle?
    MOV  rxDivide, #Baud1x0 ' (2)  yes, reload bit timer
    DEC  rxCount        ' (1)  update bit count
    SZ   ' (1/2)
    RR   rxBuf          ' (1)  position for next
bit
    SNZ  ' (1/2)
    SETB rxReady       ' (1)  alert foreground

RX_Done:
    BANK 0              ' (1)
    ENDASM

' -----
' Dimmer Processing
' -----
```

## The Nuts & Volts of BASIC Stamps 2007

## Column #146: Dimming the Lights Fantastic

```
Check_Dimmer_Tix:
ASM
    BANK  dimmerTix                ' (1)
    INC   dimmerTix                ' (1)  update dimmer divider
    CJB   dimmerTix, #5, Dimmer Done ' (2/4) ready for service?
    CLR   dimmerTix                ' (1)  yes, reset and go

' Dimmer service runs every 32.55 uS (256 levels x 120 Hz)
' -- inspired by code from Phil Short
,
Dimmer Service:
    BANK  dimmer                    ' (1)
    JNB   ZCross, Update_Triacs     ' (2/4) skip if not at ZC

Zero Cross:
    CLR   Lamps                    ' (1)  all gates off
    MOV   acc1, chan1              ' (2)          reset   dimming
accumulators
    MOV   acc2, chan2              ' (2)
    MOV   acc3, chan3              ' (2)
    MOV   acc4, chan4              ' (2)
    MOV   acc5, chan5              ' (2)
    MOV   acc6, chan6              ' (2)
    MOV   acc7, chan7              ' (2)
    MOV   acc8, chan8              ' (2)

Update Triacs:
    INC   acc1                    ' (1)  update accumulator
    SNZ                                     ' (1/2) rollover?

    SETB  Lamp1                    ' (1)  yes, triac on
    INC   acc2                    ' (1)
    SNZ                                     ' (1/2)
    SETB  Lamp2                    ' (1)
    INC   acc3                    ' (1)
    SNZ                                     ' (1/2)
    SETB  Lamp3                    ' (1)
    INC   acc4                    ' (1)
    SNZ                                     ' (1/2)
    SETB  Lamp4                    ' (1)
    INC   acc5                    ' (1)
    SNZ                                     ' (1/2)
    SETB  Lamp5                    ' (1)
    INC   acc6                    ' (1)
    SNZ                                     ' (1/2)
    SETB  Lamp6                    ' (1)
    INC   acc7                    ' (1)
    SNZ                                     ' (1/2)
    SETB  Lamp7                    ' (1)
    INC   acc8                    ' (1)
    SNZ                                     ' (1/2)
```

## Column #146: Dimming the Lights Fantastic

```
    SETB  Lamp8                                ' (1)

Dimmer Done:
    BANK  0                                    ' (1)
    ENDASM

    RETURNINT

' =====
' PROGRAM Main
' =====

' -----
' Subroutine Declarations
' -----

DELAY_MS      SUB    1, 2                      ' delay in milliseconds
RX BYTE       FUNC  1, 0                      ' receive a byte

' -----
' Program Code
' -----

Main:
    breakTmr = 0                              ' arm the break timer
    hasBreak = No
    DO WHILE hasBreak = No                    ' wait for break
    LOOP

Flush Uart:
    rxCount = 0
    rxReady = No

Align_Packet:
    idx = 0
    idx.0 = ~Addr.3                            ' read address
    idx.1 = ~Addr.2
    idx.2 = ~Addr.1
    idx.3 = ~Addr.0
    idx = idx << 3                            ' bytes to skip (idx * 8)

Skip Bytes:
    DO WHILE idx > 0
        tmpB1 = RX BYTE                        ' flush from stream
        IF hasBreak = Yes THEN Main           ' abort on packet break
        DEC idx
    LOOP
```

## The Nuts & Volts of BASIC Stamps 2007

```

Get_Levels:                                ' rx levels for this board
  FOR idx = 0 TO 7
    IF hasBreak = Yes THEN Main
      dimmer(idx) = RX BYTE
  NEXT
  GOTO Main

' -----
' Subroutine Code
' -----

' Use: DELAY_MS ms
' -- delay in ~milliseconds; uses ISR
' -- ideal ISR count is 153.8

SUB DELAY MS
  IF __PARAMCNT = 1 THEN
    tmpW1 = __PARAM1                    ' save byte value
  ELSE
    tmpW1 = WPARAM12                   ' save word value
  ENDIF
  DO WHILE tmpW1 > 0
    tmpB1 = 154                         ' time left?
    DO WHILE tmpB1 > 0
      \ CLRB isrTic                     ' (re)load 1 ms timer
      \ JNB isrTic, $                   ' clear ISR flag
      DEC tmpB1                          ' wait for next flag
      DEC tmpB1                          ' update 1 ms timer
    LOOP
    DEC tmpW1                            ' update delay timer
  LOOP
ENDSUB

' -----

' Use: result = RX_BYTE
' -- if no byte available routine will wait

FUNC RX BYTE
  IF rxReady = 0 THEN RX BYTE          ' wait
  tmpB1 = rxBuf                        ' get from buffer
  rxReady = 0                          ' allow new RX
RETURN tmpB1
ENDFUNC

' =====
' User Data
' =====

```

**Column #146: Dimming the Lights Fantastic**

**The Nuts & Volts of BASIC Stamps 2007**