Column #61, May 2000 by Jon Williams:

Stamping Myself Into Better Shape

It's hard to believe, but the BASIC Stamp has been around for almost seven years now. While this column isn't quite that old, it's been around awhile too and the three of us that have written it have delved into some reasonably sophisticated projects. Parallax has made terrific efforts in the way of getting Stamps into the hands of youngsters through their "Stamps in Class" program. What this means is a whole new crop of young Stampers. This month we're going to take a step back in sophistication and focus on the process of program development. It's a simple project and yet, might even teach the "old dogs" a few new tricks.

One of the best things about doing electronics as a hobby or profession is the ability to design something useful that doesn't currently exist. Case in point:

Some of you know that I'm a part-time actor. Yes, really, an actor – I have an agent, go on auditions and even get hired from time-to-time. I was recently lamenting to a friend about some great auditions I'd had that didn't result in bookings. It is the worst part of being an actor: the unexplained rejections. My friend reminded me that – especially in commercials -- most of the time it's about appearance, and that almost everyone being called in to audition can act. It made me think long and hard. I've spent the last two and a half years training with the best acting coaches in Dallas and, sad to say, have not dealt much with my appearance. I guess I could blame my advancing age for my advancing

waistline, but the fact is I really should make an effort to drop a few pounds. I just saw myself on tape from my last stage appearance and was not happy.

I study with an actor named Tim who is in great physical shape. On a class break, I approached Tim about his exercise regimen, expecting to hear about how he spends a couple hours a day in the gym and eats next to nothing. Not so. Tim told me how he entered a 12-week contest and dropped an amazing amount of weight and completely transformed his physique. He even showed me his driver's license photo to prove it. He had made an amazing transformation and had spent less that three hours per week in the gym! Hey, I can live with that.

Well, I don't really want to enter a contest, I just want to drop 10 or 15 pounds and improve my physical appearance. As luck would have it, I found a book that detailed the diet and exercise regimen that Tim follows. To my very happy surprise, the diet and exercise plan is not difficult, yet it is regimented and needs to be followed with some discipline.

Specific to the exercise plan is the timing. While the aerobic part of the plan calls for only 20 minutes of exercise every other day, it must be done is a specific manner: four cycles of five minutes each, with each cycle divided into four stages (two minutes, one minute, one minute, then one minute).

I turned to my cool digital wristwatch. Darn! The timer has only three stages and between the overhead TVs, the radio blaring and the other treadmills, I can't hear the watch beeping in the noisy environment of they gym, anyway. Sounds like a good opportunity to build a custom circuit. Sounds like a great opportunity to using the BASIC Stamp! So, I want a little exercise timer that will time four cycles of five stages and give me both visual indicators for use in the noisy gym and an audible indicator for those days that I'm on the jogging trail and the unit is strapped to my arm.

Before we jump in, let's talk a bit about the "process." What I mean by this is the steps that will follow to develop our project. Don't make the mistake thinking that you can just jump right into small projects without planning them – like dogs, small projects can bite you as badly as big ones.

Here's a good process to use for Stamp project development:

- Define the project/product
- Design the hardware
- Write test code to verify hardware design
- Write software
- Test software to requirements
- Field test project
- Refine software and re-test

The first step, "define the project," is the trickiest and causes the most amount of trouble because it is usually taken for granted. I'm sure you've heard the phrase "Plan you work, work your plan." That's what step 1 is all about. You'll save a lot of time and, especially if you enter the professional ranks, a lot of development money if you commit to step 1.

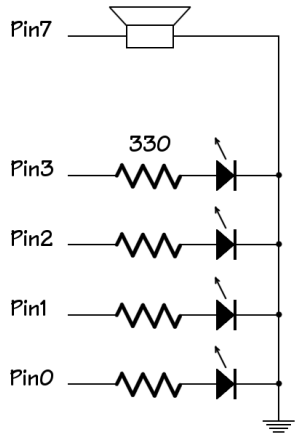
How do we do this? The answer is simple and yet, it's not always easy: we must fully understand our customer's wants and needs. So take the time to do this step well. Ask a lot of questions. Sometimes the customer will be you. That's okay and doesn't excuse you from the step. And if the customer is someone else, especially if that someone will be paying you for the product, you really need to be diligent. If you don't, you may find yourself delivering a product that is missing features or, worst of all, is nothing close to what your customer actually wants.

Okay, I've already described our little exercise timer and we're just going to leave the definition as is for the moment. Let's move on to the hardware (step 2).

Our timer needs four stages, so we'll use an LED for each. Since we have a little bit of Stamp experience, we'll connect the LEDs to the lower bits of the output and use positive logic (output high will light the LED). This will be fine since we're only going to turn on only one LED at a time. We can get a little more current out of the Stamp if we use inverted logic (output low turns on LED), but this makes the code a little more difficult to read. The last thing we need is something to make sound. A small piezo speaker will take care of that nicely. See Figure 61.1 for the schematic of the exercise timer.

With the hardware designed, we need to test it. This is especially important there is a lot of hardware interacting with the Stamp. While that's not the case with this project, we still need to test it and we'll use this opportunity to develop a little bit of the code for the final project.

Figure 61.1: Exercise timer schematic



Program Listing 61.1 is the test code. To some, this code may look a little more refined that what one would generally consider "test" code, but it's best to start with neat code from the start. That way, we're not cleaning up our test code to put into production; it's usable as is.

For you beginners, I can't stress enough the importance of using SYMBOLs. Using SYMBOLs will save you time by making your code somewhat self-documenting. This allows you to focus on the code instead of the comments. And don't forget that PBASIC allows you to assign more than one SYMBOL to the same variable (this is called aliasing). You can do this when you're running out of variables and want to use the same one in different parts of the program. Just keep in mind that PBASIC variables are global and you'll want to make sure that using a variable in one part of a program under a given alias does not interfere with it's operation in another part of the program. Here again, planning will keep you out of these sticky spots.

The test program is pretty basic. We want to make sure that the LEDs work and that the beeper sound is pleasing. To that end, we've decided to structure the code like a (shortened) exercise cycle. One thing that I know I'm going to need is some sort of delay routine. We can't get away with a simple PAUSE command because the first stage is two minutes long. Since the maximum PAUSE value is 65535 – just over a minute – it won't work.

We could choose to use PAUSE 60000 twice for the first stage, but this doesn't give me much flexibility or re-use in this project or others. A delay routine that works in seconds would be a little more flexible and easier to apply elsewhere. In the Subroutines section you'll find "DlySec." This works by setting the variable "secs" to the value we want and calling the routine with GOSUB. By using byte-sized variables this routine will give us a delay of four minutes, 15 seconds (255 seconds). If you want to modify this to run longer, you'll need to change both "secs" and "loops" to word-sized variables.

Okay, the LEDs work and the beeper sounds great, so let's start the production code (step 4). Program Listing 61.2 is our first attempt. You see that it's really not too much of a change from the test code. In fact, the only thing necessary to get us from the test code to meeting the basic specification was to create a loop in the main body of code so that the program will run four cycles.

The code works – great! Now we could stop here, but what we've noticed is that there's a big chunk of code that has redundant sections that we may be able to refine. If we look at the operation of each stage, it goes like this: turn on the stage LED, sound the beeper, time the stage. Yeah, we can clean this up a bit.

Take a look at Program Listing 61.3. What we've done is added an inner loop to handle the four stages of each cycle. We're able to deal with the discrete differences between each stage by using LOOKUP to get the values we need for the LED output and for the stage timing. Notice that our stage loop is zero-based (goes from zero-to-three instead of one-to-four). This is necessary to conform to the requirements of LOOKUP. The first element in the LOOKUP table corresponds to the control variable value of zero.

All right, time for testing. Everything seems to be working fine and we even used less code space by adding the stage loop. That's good, it means there's more room for future features. Then something happened. I got up to get a drink of water and while walking away, the beeper sounded. Cool, the next stage. But what stage? While I'm out on the jogging trail I don't want to think about where I am – I just want to be told what to do next.

What we need to do is have a distinct audio indication for each stage, just as each stage has its own LED. Take a look at Listing 61.4. What we've done is replaced the single SOUND command that's embedded in our loop with a call to a subroutine named, appropriately enough, "Beep." Beep takes advantage of another great PBASIC command, BRANCH. We use BRANCH to route the routine to the appropriate sound commands. Stage one uses one low-frequency beep, stage two uses two low-frequency

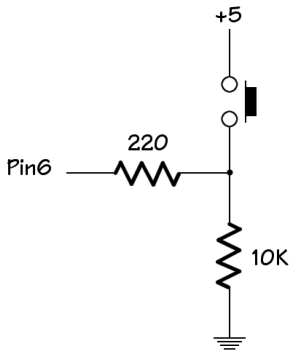
beeps, stage three uses one high-frequency beep and stage four uses two high-frequency beeps.

That's much better. Now it's time to move from the breadboard to a device that can be "field tested" (step 6) It's one thing to conform to specifications in the lab, but it's in actual use where we really find out how well we've done are job.

Okay, I put on some gym shorts, laced up my favorite jogging shoes and hit the trail. So far, so good. I'm thinking, "Yeah, baby, I can feel the pounds melting away." Then, just as I was so proud of myself, I tripped over an untied shoelace. To make matters worse, the lace breaks as I'm re-tying it and then the beeper indicates it's time for the next stage. Shoot! This isn't working out – I need a Hold switch on the timer.

To be fair, we can't always foresee these problems in the design stage – that's why we do field testing. That said, we would minimize the chances of major problems arising during field test by asking a lot of questions up front; especially questions on how the device will be used. Field testing, as we've just seen, will catch all the final "gotchas."

Figure 61.2: Hold switch hardware



Adding the hardware for the Hold switch is the easy part (Figure 61.2); the software to support it is going to require a little more work. The reason is that we've keep things very simplistic. I'm not criticizing what we've done so far because it does work. The simplicity, however, does not support much in the way of flexibility and now we have to take a step back and re-design our code. Now you see why it's so important to know – to the extent we can – all of your project requirements up front.

Our Hold switch will work by suspending the current stage until pressed again. The device will tell us it's on Hold by flashing the current stage LED. Finally, we'll start the program in Hold mode since it's just a bit easier to press the Hold button than to flip the power switch when we want to get started.

You may remember from last month that I suggested we could give our programs a lot of flexibility by breaking them up into small chunks and using a "task switcher" or "time slicing" approach. Since the concept as it applies to this project may not be immediately obvious, let's just jump into the final code and see how we got there. Once you understand the concept, you should find it reasonably easy to apply to your time-oriented programs.

What you'll immediately notice is that they delay routine is gone and we've added quite a few variables. The approach of this program takes advantage of the fact that most programs run in a loop. What we're going to do is keep the loop simple and somewhat constant, allowing us to use it as a time base.

Did you know that the first PCs didn't have real-time-clock chips? They kept time, at their lowest level, by keeping a tic counter. Tics were then converted into the appropriate time format. That's what we're going to do. Our code will run in a continuos loop and each pass through the loop will cause the tic counter to be updated. Actual loop timing is a bit of a trade and some projects will require experimentation to get the best results. We have an additional requirement: to monitor and debounce the Hold switch. Yes, we could just use the PBASIC BUTTON command, but it's not my favorite and can be a little confusing. What we'll find is that loop-based architecture of our code will support monitoring and debouncing the switch with fairly simple code.

For the moment, we'll arbitrarily decide to run our loop at 20 times per second, or 50 milliseconds per loop iteration. Since our stage delay is in seconds, we'll want to count 20 tics (loops) and then update a seconds counter. When the seconds counter reaches the stage timing value, we'll move on to the next stage.

Take a look at the main body of code. The beginning of each stage is noted with the stage LED lit and an audio indication. This will only occur when our seconds timer (timing the current stage) and the tic counter (time base for seconds) are zero. We also need to load the stage timing when we start a new stage.

At the heart of our loop is the PAUSE command that pads it to 50 milliseconds. We need this because even the Stamp 1 can zip through its code pretty quickly. Now you'll notice that the SYMBOL called "LoopTm" is not set at 50 (it's 46 in the final code). It did, indeed, start at 50 and was fine-tuned during testing. I'll explain how at the end.

Now we get to the switch monitoring and debouncing. The way this will work is by incrementing a counter whenever the switch is pressed. If the switch is released, the counter is cleared. When the counter reaches the threshold that indicates a valid switch press, some action will be taken.

By taking advantage of the Stamp's architecture and the fact that pins can be treated like variables, the switch variable updating or clearing can be done with one line of code. This works because a pin is a bit-sized variable and can have a value of zero or one. Our hardware design supports the code design by pulling the Hold pin low (value of zero) when the switch is released. Pressing the switch brings the pin up to five volts, which is read as one. So, when the switch is pressed, we will add one to our counter and then multiply by one (which has no affect on the counter). When the switch is released, we add zero (no change) but then multiply by zero, which immediately clears the counter. This trick saves a potentially tedious IF-THEN construct in the code.

The next line checks the switch value. If it reaches five (corresponding to 250-millisecond press; the reason our loop time is less than one second) the code will jump to the routine called "OnHold." This routine flashes the current stage LED by setting the Dirs register to inputs then back to outputs. The same switch monitoring and debouncing technique is used to release the timer from hold.

The final part of the loop is testing and updating the various timers and counters. Once again we save code by taking advantage of Stamp math and by adopting zero-based counters. By using the Modulus (//) operator (remainder of a division) and counting from zero to our max value minus one, we considerably simplify the code. Any number MOD itself is zero. By using this technique a value of zero indicates a rollover and the need to update our next level counter.

So you see that in the section called "Test" that the counters are tested from the inside out. First we check the tic counter. If it rolls over to zero, a second has elapsed; otherwise we go back to the main loop. The same process works for the seconds counter, the stage counter and the cycles counter. When the cycle counter rolls over to zero we've completed all for cycles and are done. I'm sure that you can see how easy it would be to synthesize a real time clock using this technique to generate seconds, minutes and hours.

The last step is testing and fine-tuning. We'll find that with a "LoopTm" value of 50, the timer will run long. That shouldn't come as a surprise since we know that the code itself will take some time to run. Here's how to determine the final value for "LoopTm." Start a stopwatch and time the first stage. We get a value of about 129 seconds – nine seconds too long. If we take 120 (the desired time) and divide by 129 (tested time) we get 0.93. Now we multiply 0.93 by 50 (loop pad) and get 46.5. We'll round down to 46 for our "LoopTm" value. We run the stopwatch test again and find that it's dead on. Perfect – time to hit the jogging trail.

Even if this timer project is of no value to you, the technique used in the final program can be when you apply it to your projects. This timing technique, especially when combined with task-switching techniques is terrific for Stamp-based robotics. Give it a try, because we will be using it more frequently in future articles. Until then, happy Stamping.

Column #61: Stamping Myself into Better Shape

```
' Stamp Applications - May 2000
' Program Listing 61.1


' ----[ Title ]-----------------------------------------------------------
'
' File...... XTIMER1.BAS
' Purpose... Exercise Timer - Hardware test code
' Author.... Jon Williams


' ----[ Program Description ]---------------------------------------------
'
' Hardware test code for exercise timer project


' ----[ I/O Pins ]--------------------------------------------------------
'
SYMBOL  BprPin = 7


' ----[ Constants ]-------------------------------------------------------
'
SYMBOL  BprTone = 75
SYMBOL  BprLen = 16                         ' 0.192 secs


' ----[ Variables ]-------------------------------------------------------
'
SYMBOL  secs = B2
SYMBOL  loops = B3


' ----[ Initialization ]--------------------------------------------------
'
Init:   Pins = %00000000                    ' LEDs off to start
        Dirs = %10001111                    ' LEDs and Piezo outs


' ----[ Main Code ]-------------------------------------------------------
'
Main:   Pins = %0001                        ' stage 1 LED on
        SOUND BprPin,(BprTone, BprLen)      ' sound start of stage
        secs = 120                          ' 2-minute stage
        GOSUB DlySec

        Pins = %0010                        ' stage 2
        SOUND BprPin,(BprTone, BprLen)
        secs = 60
        GOSUB DlySec
```

```
        Pins = %0100                        ' stage 3
        SOUND BprPin,(BprTone, BprLen)
        GOSUB DlySec

        Pins = %1000                        ' stage 4
        SOUND BprPin,(BprTone, BprLen)
        GOSUB DlySec

Done:   Pins = %0000                        ' LEDs off
        SOUND BprPin,(0,12,50,12,75,12,110,12)' sound end

        END                                 ' of program
' ----[ Subroutines ]------------------------------------------------------
DlySec: FOR loops = 1 TO secs
          ' PAUSE 1000                      ' pause 1 second
          PAUSE 10                          ' quick pause for testing
        NEXT
        RETURN
```

```
' Stamp Applications - May 2000
' Program Listing 61.2


' ----[ Title ]-------------------------------------------------------------
'
' File...... XTIMER2a.BAS
' Purpose... Exercise Timer - Version 2 (4 cycles, full time)
' Author.... Jon Williams


' ----[ Program Description ]-----------------------------------------------
'
' This program serves as a multi-stage exercise timer.  The program runs
' four cycles of four stages, with an LED and audio indication for each
' stage.


' ----[ I/O Pins ]----------------------------------------------------------
'
SYMBOL  BprPin = 7


' ----[ Constants ]---------------------------------------------------------
'
SYMBOL  BprTone = 75
SYMBOL  BprLen = 16                         ' 0.192 secs


' ----[ Variables ]---------------------------------------------------------
'
```

```
SYMBOL  cycle = B2
SYMBOL  secs= B3                            ' stage timing (minutes)
SYMBOL  loops = B4                          ' counter for long delay


' ----[ Initialization ]----------------------------------------------------
'
Init:   Pins = %00000000                    ' LEDs off to start
        Dirs = %10001111                    ' LEDs and Piezo outs


' ----[ Main Code ]---------------------------------------------------------
'
Main:   FOR cycle = 1 TO 4

          Pins = %0001                      ' stage 1 LED on
          SOUND BprPin,(BprTone, BprLen)    ' sound start of stage
          secs = 120                        ' 2-minute stage
          GOSUB DlySec

          Pins = %0010                      ' stage 2
          SOUND BprPin,(BprTone, BprLen)
          secs = 60
          GOSUB DlySec

          Pins = %0100                      ' stage 3
          SOUND BprPin,(BprTone, BprLen)
          GOSUB DlySec

          Pins = %1000                      ' stage 4
          SOUND BprPin,(BprTone, BprLen)
          GOSUB DlySec

        NEXT ' cycle

Done:   Pins = %0000                        ' LEDs off
        SOUND BprPin,(0,12,50,12,75,12,110,12) ' sound end

        END ' of program


' ----[ Subroutines ]-------------------------------------------------------
'
DlySec: FOR loops = 1 TO secs
          PAUSE 1000                        ' pause 1 second
          ' PAUSE 10                        ' quick pause for testing
        NEXT
        RETURN
```

```
' Stamp Applications - May 2000
' Program Listing 61.3


' ----[ Title ]-----------------------------------------------------------
'
' File..... XTIMER2b.BAS
' Purpose... Exercise Timer - Version 2 (refined main loop)
' Author.... Jon Williams


' ----[ Program Description ]---------------------------------------------
'
' This program serves as a multi-stage exercise timer.  The program runs
' four cycles of four stages, with an LED and audio indication for each
' stage.


' ----[ I/O Pins ]--------------------------------------------------------
'
SYMBOL  BprPin = 7


' ----[ Constants ]-------------------------------------------------------
'
SYMBOL  BprTone = 75
SYMBOL  BprLen = 16                         ' 0.192 secs


' ----[ Variables ]-------------------------------------------------------
'
SYMBOL  cycle = B2                          ' cycles counter
SYMBOL  stage = B3                          ' stage counter
SYMBOL  secs= B4                            ' stage timing (seconds)
SYMBOL  loops = B5                          ' counter for long delay


' ----[ Initialization ]--------------------------------------------------
'
Init:   Pins = %00000000                    ' LEDs off to start
        Dirs = %10001111                    ' LEDs and Piezo outs


' ----[ Main Code ]-------------------------------------------------------
'
Main:   FOR cycle = 1 TO 4
          FOR stage = 0 TO 3
            LOOKUP stage,(%0001,%0010,%0100,%1000),Pins
            SOUND BprPin,(BprTone, BprLen)
            LOOKUP stage,(120,60,60,60),secs
```

```
           GOSUB DlySec
         NEXT ' stage
       NEXT ' cycle

Done:  Pins = %0000                          ' LEDs off
       SOUND BprPin,(0,12,50,12,75,12,110,12) ' sound end

       END ' of program


' ----[ Subroutines ]-----------------------------------------------------
'
DlySec: FOR loops = 1 TO secs
         PAUSE 1000                          ' pause 1 second
         ' PAUSE 10                          ' quick pause for testing
       NEXT
       RETURN
```

```
' Stamp Applications - May 2000
' Program Listing 61.4


' ----[ Title ]-----------------------------------------------------------
'
' File...... XTIMER2c.BAS
' Purpose... Exercise Timer - Version 2 - Final
' Author.... Jon Williams


' ----[ Program Description ]---------------------------------------------
'
' This program serves as a multi-stage exercise timer.  The program runs
' four cycles of four stages, with an LED and audio indication for each
' stage.


' ----[ I/O Pins ]--------------------------------------------------------
'
SYMBOL  BprPin = 7


' ----[ Constants ]-------------------------------------------------------
'
SYMBOL  BprTone = 75
SYMBOL  BprLen = 16                          ' 0.192 secs


' ----[ Variables ]-------------------------------------------------------
'
SYMBOL  cycle = B2                           ' cycles counter
```

```
SYMBOL  stage = B3                          ' stage counter
SYMBOL  secs= B4                            ' stage timing (seconds)
SYMBOL  loops = B5                          ' counter for long delay


' ----[ Initialization ]----------------------------------------------
'
Init:   Pins = %00000000                    ' LEDs off to start
        Dirs = %10001111                    ' LEDs and Piezo outs


' ----[ Main Code ]----------------------------------------------------
'
Main:   FOR cycle = 1 TO 4
          FOR stage = 0 TO 3
            LOOKUP stage,(%0001,%0010,%0100,%1000),Pins
            GOSUB Beep
            LOOKUP stage,(120,60,60,60),secs
            GOSUB DlySec
          NEXT ' stage
        NEXT ' cycle

Done:   Pins = %0000                        ' LEDs off
        SOUND BprPin,(0,12,50,12,75,12,110,12) ' sound end

        END ' of program


' ----[ Subroutines ]--------------------------------------------------
'
Beep:   BRANCH stage,(Beep1,Beep2,Beep3,Beep4)
Beep1:  SOUND BprPin,(50,12)
        GOTO BeepX
Beep2:  SOUND BprPin,(50,12,0,8,50,12)
        GOTO BeepX
Beep3:  SOUND BprPin,(90,16)
        GOTO BeepX
Beep4:  SOUND BprPin,(100,16,0,8,100,16)
BeepX:  RETURN

DlySec: FOR loops = 1 TO secs
          PAUSE 1000                        ' pause 1 second
          ' PAUSE 10                        ' quick pause for testing
        NEXT
        RETURN
```

```
' Stamp Applications - May 2000
' Program Listing 61.5

' ----[ Title ]---------------------------------------------------------
'
' File..... XTIMER3.BAS
' Purpose... Exercise Timer - Version 3 - "Time Slicer"
' Author.... Jon Williams


' ----[ Program Description ]-------------------------------------------
'
' This program serves as a multi-stage exercise timer.  The program runs
' four cycles of four stages, with an LED and audio indication for each
' stage.
'
' The timer includes a pause switch so that a stage can be interrupted and
' restarted.  The timer begins in "Hold" mode.  Pressing the Hold switch
' starts the timer


' ----[ I/O Pins ]------------------------------------------------------
'
SYMBOL  BprPin = 7                          ' beeper pin
SYMBOL  SWin = Pin6                         ' Hold switch input


' ----[ Constants ]-----------------------------------------------------
'
SYMBOL  LoopTm = 46                         ' 1/20 second loop (tuned)
SYMBOL  TixMax = 20                         ' 20 x 50 ms = 1 second
SYMBOL  SwMax = 5                           ' switch debounce value

SYMBOL  Up = 0                              ' Hold switch released
SYMBOL  Down = 1                            ' Hold switch pressed


' ----[ Variables ]-----------------------------------------------------
'
SYMBOL  cycle = B2                          ' 4 cycles per workout
SYMBOL  stage = B3                          ' 4 stages per cycle
SYMBOL  tix = B4                            ' timing counter
SYMBOL  secs = B5                           ' stage seconds counter
SYMBOL  sMax = B6                           ' max time for stage
SYMBOL  swtch = B7                          ' switch debounce


' ----[ Initialization ]------------------------------------------------
'
Init:   Pins = %00000001                    ' light stage one at start
```

```
        Dirs = %10001111                        ' LEDs and Piezo outs

        cycle = 0                               ' first cycle
        stage = 0                               ' first stage
        tix = 0                                 ' clear tix timer
        secs = 0                                ' clear seconds

        GOTO OnHold                             ' start in Hold mode


' ----[ Main Code ]-----------------------------------------------------
'
Main:   IF secs > 0 THEN Pad                    ' beep only when secs and
        IF tix > 0 THEN Pad                     '   tix are zero
        LOOKUP stage,(%0001,%0010,%0100,%1000),Pins  ' stage LED
        GOSUB Beep
        LOOKUP stage,(120,60,60,60),sMax        ' get stage timing (secs)

Pad:    PAUSE LoopTm                            ' pad the loop for timing

ChkSw:  swtch = swtch + SWin * SWin             ' check pause switch
        IF swtch >= SwMax THEN OnHold           ' debounce; call Hold if
                                                '  held down

Test:   tix = tix + 1 // TixMax                 ' increment with rollover
        IF tix > 0 THEN Main                    ' still in same second
        secs = secs + 1 // sMax                 ' increment seconds
        IF secs > 0 THEN Main                   ' still in stage
        stage = stage + 1 // 4                  ' increment stage
        IF stage > 0 THEN Main                  ' still in cycle
        cycle = cycle + 1 // 4                  ' increment cycle
        IF cycle > 0 THEN Main                  ' still running

Done:   Pins = %0000                            ' LEDs off
        SOUND BprPin,(0,12,50,12,75,12,110,12) ' sound end

        END ' of program


' ----[ Subroutines ]---------------------------------------------------
'
Beep:   BRANCH stage,(Beep1,Beep2,Beep3,Beep4)
Beep1:  SOUND BprPin,(50,16)
        GOTO BeepX
Beep2:  SOUND BprPin,(50,16,0,8,50,16)
        GOTO BeepX
Beep3:  SOUND BprPin,(90,16)
        GOTO BeepX
Beep4:  SOUND BprPin,(90,16,0,8,90,16)
        GOTO BeepX
BeepX:  RETURN
```

```
OnHold: IF SWin = Down THEN OnHold          ' wait for release
        swtch = 0                           ' clear switch check
Hold1:  Dirs = %00000000                    ' turn of LED
        PAUSE 65
        Dirs = %10001111                    ' back on
        PAUSE 65
        swtch = swtch + SWin * SWin         ' check switch
        IF swtch < 2 THEN Hold1             ' debounce
        swtch = 0                           ' reset timers/counters
        tix = 0
        secs = 0
        GOTO Main
```