

# CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

## FEATURE ARTICLE

Jeff Loeliger

# RC Servo Control Via TPU

If you've ever wanted to control RC servos without any additional hardware, then pay attention to this project because that's just what Jeff has done. By designing a time processor unit (TPU) function, he provides an easy-to-use interface and puts some of the fun back into servo control.

**W**hen dabbling in robotics, one of the hardest things encountered is obtaining the motors and geartrains. One solution is to use RC servos—they are cheap, lightweight, hard wearing, come in many sizes, and have a standard interface. The drawback is that they are relatively hard to control.

There are a few options open for controlling servos. You can use the main micro (obviously takes up valuable CPU time), a separate control board (more expensive and cumbersome), or you can use a custom chip (again, more expensive).

This article covers implementing a fourth option that involves no extra cost and no loading to the CPU, and provides control of up to 16 servos.

### PROBLEM DEFINITION

Servos have a three-wire interface: ground, power, and control. The input to the control line is a pulse-code modulated signal from which all servo timings and positions are derived. All

servos have their own limits but convention states that applying a 1.5-ms signal holds the servo in neutral, a 1-ms signal turns it full counter-clockwise, and a 2-ms signal results in a full clockwise turn (see Figure 1). The signal must be repeated no less than every 30–50 ms or the servo may start jittering and finally stop driving the output. The result would be the loss of active hold on the desired position. The use of timer hardware and software algorithms is therefore essential to allow the accurate control of timed events.

As I mentioned earlier, there are three options currently available for servo timing control. The first method is the main microcontroller. When controlling the servo with software alone, it's often hard to obtain an accurate output of the control signal. If the CPU is overloaded, the signal may be less accurate which causes the servo to jitter. This form of control also places extra loading on the CPU. Timer hardware can be used on the main microcontroller to eliminate jitter. However, the number of channels available is usually limited.

The second method, using an external board for servo control, adds to the cost of the system and takes up valuable space (a commodity not usually in abundance in robotic systems).

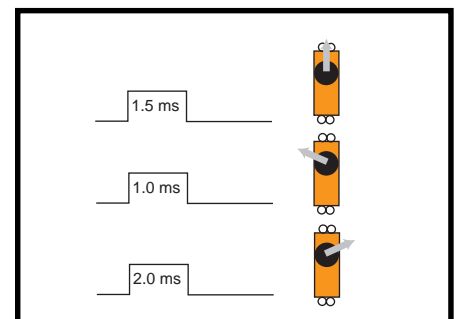


Figure 1—The control signal for a servo should be repeated every 30–50 ms. By convention, 1.5 ms signal is centered.

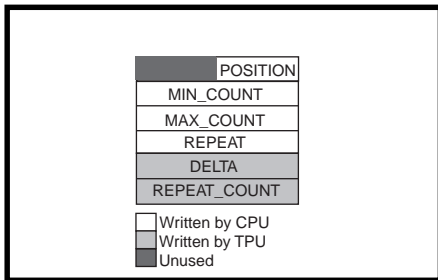


Figure 2—Each TPU channel has its own parameter and this shows how they are laid out for the servo function.

Servo control with an external board requires a serial signal via a UART interface. The main micro-controller must therefore have a spare UART interface to talk to the servo controller. Coordination problems can also arise because the speed of the UART interface determines how fast the CPU can update the servo positions.

The third and final option is the custom chip. This form of control has the same UART issues as described in the external board solution. Although less of an issue on size, it can significantly add to the overall system cost.

## THE ANSWER IS...

The solution proposed here is to use the Motorola time processor unit (TPU) to generate the necessary control signals for the servo. The TPU is an intelligent timing coprocessor module. It has its own program and data memory and can generate complex waveforms without CPU intervention. It is available on many of Motorola's 16- and 32-bit micro-controllers.

Using the TPU eliminates all the issues encountered by the current servo control options available. With this solution, generating the control signal causes no loading on the CPU therefore freeing it up to handle CPU-critical functions. The TPU allows a simple interface to the servo, making control easy and, as the output signal is always accurate, jitter is not a problem. No external hardware is required, eliminating the extra cost and space required with the custom chip and external board solutions currently available. Probably the most important feature of this solution is the TPU's ability to control up to 16 servos simultaneously using its 16 I/O

channels. The Motorola MPC555 has two TPU3 (for information on the differences between TPU, TPU2, and TPU3, see this sidebar) modules onboard and can therefore push this even further by controlling 32 servos at the same time.

## SOLUTION DESCRIPTION

The servo function has four parameters that need to be configured by the host CPU (see Figure 2). The first parameter, POSITION, is an 8-bit value that represents the desired position of the output signal. This is the only parameter that needs to be changed while the function is actually running. A change performed by the user will vary the position of the servo. The remaining three parameters define and limit the output waveform and are in counts referenced by the selected timebase.

There are two timebases on the TPU that can be selected by the user. The MIN\_COUNT parameter defines the minimum length of the control signal with MAX\_COUNT defining the maximum length. Both of these parameters must be less than \$8000 and the MIN\_COUNT must be set lower than the MAX\_COUNT.

The range of movement of the servo can be limited by using MIN\_COUNT and MAX\_COUNT values that are less than the actual minimum and maximum values for the servo. By decreasing the range, the

resolution is increased. The REPEAT parameter specifies the amount of time between the rising edges of the control signal. The user specifies the number of times \$8000 timer counts should be repeated during the output delay. The time is then calculated as  $Delay = REPEAT \times \$8000$ . There are two additional parameters, DELTA and REPEAT\_COUNT, used by the function but not changed by the user.

There are two host-sequence bits, HSQ0 and HSQ1, that control the configuration of the servo function. HSQ0 is used to select one of the two timebases in the TPU—0 for timebase 1 (TCR1) and 1 for timebase 2 (TCR2). HSQ1 is used to choose between normal and reversed servo operation. Normally, a position value of 0 represents the minimum time for the output signal with \$FF representing the maximum time. When HSQ1 is set, the operation is reversed with 0 representing the maximum time and \$FF the minimum. This arrangement allows greater flexibility and is a quick-fix solution should the servo turn out to travel in the opposite direction to that planned. In addition, the servo function uses two host service requests—HSR%11 and HSR%01. HSR%11 initializes the function and HSR%01 causes the immediate update of the servo position.

## DESIGNING FUNCTIONS

The best way to design TPU functions is through state diagrams. The TPU is event driven, which correlates well to the states. This function has five states that initialize and control the output of the servo control signal. The state diagram and associated control waveform is given in Figures 3 and 4. Table 1 outlines the actions of each state.

State 0 is the first state executed. It's responsible for initializing the whole function and scheduling the rising edge match. After the parameter RAM in the TPU is set up, the host CPU issues a host service request (HSR%11) to the TPU, which causes state 0 to be executed. State 2 generates the rising edge of the output control signal and calculates when the falling edge should happen. State 3 is entered when the falling edge is gener-

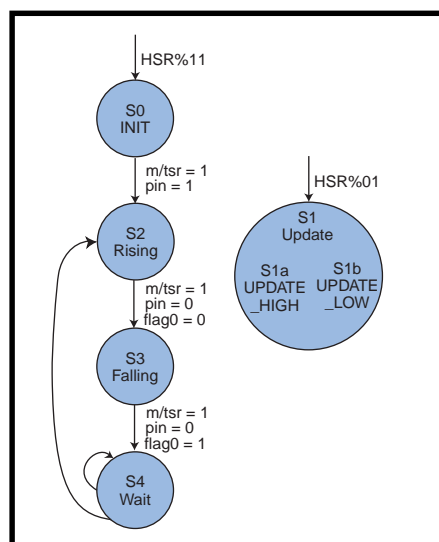


Figure 3—The main flow is for normal operation of the function and state 1 is used for immediate updates.

ated. It starts the long delay that is held by state 4 until the next control signal is generated.

In addition to the basic control states, there is an Immediate Update state (state 1). Normally the output signal is only updated every 20–30 ms. State 1 gives the option of updating it sooner as long as the control waveform is still high. If the TPU is currently generating the active portion of the output signal, updating the POSI-

TION parameter and issuing a host service request (HSR%01) will cause the function to change the output signal value immediately.

State 2 must perform a multiply-and-divide calculation to determine the output signal. Because the TPU does not have a multiply instruction, hardware is used to help perform a shift-and-add multiply algorithm. To calculate the output

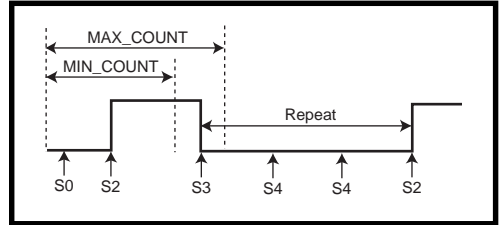


Figure 4—Here you can see the relationship of the parameters and states to the output waveform.

signal, a 16 x 8 bit multiply is required, giving a 24-bit answer. The most significant 16 bits of the answer are then stored in the A register while the least significant eight bits are stored in the Shift Register (SR).

Because there are 256 output positions, a divide-by-256 would be required on the A register and Shift Register to give the final answer. This would be equivalent to eight successive right shifts so the A register can be considered to hold the final answer and the data in the Shift Register can be ignored and used merely for rounding. This method allows complex multiply-and-divide calculations to be executed in only three instructions and 10 TPU cycles. The multiply-and-divide operation, in the middle of state 2, is shown in Listing 1.

The key to creating a fast TPU function is mapping the function states to the TPU entry table efficiently. By doing this, the TPU is able to determine what state to execute without using any code. All of the states (except state 1) map directly to the TPU entry table. State 1 is instead divided into two substates (state 1a and state 1b) that are mapped directly to the entry table. States 0, 1a, and 1b are entered when a host service request (HSR) is issued.

When a match between the TPU channel's compare register and the selected TPU timebase occurs, states 2, 3, and 4 can be entered. State 2 is entered if the output pin is high after the match, with states 3 and 4 being entered if the pin is low. An internal software flag in the TPU (flag 0) then determines whether state 3 or state 4 should be entered. The TPU defines what level the output should be when a match occurs to ensure that the flow follows the state diagram.

The servo function was written for

```

<b><font face="Arial, Helvetica, sans-serif" size="2">Listing 1-State 2 of the servo function with the fast
multiply-and-divide code in the middle of the state.</font></b>

(*****)
(* ENTRY NAME:  SERVO_RISING                STATE(S) ENTERED:  S2      *)
(* STATE SIZE:  11 instructions             MAXIMUM INSTRUCTIONS EXECUTED: 17 *)
(* ACTION:      -If reverse is selected then reverse position.          *)
(*              -Setup pin for falling edge.                            *)
(*              -Clear flag0                                             *)
(*              -Calculate falling edge time.                            *)
(*              x = ( POSITION * DELTA ) / 256                            *)
(*              round answer                                             *)
(*              add to MIN_COUNT to get edge time                        *)
(*****)
%entry name = SERVO_RISING; start_address *; disable_match;
cond hsr1 = 0, hsr0 = 0, lsr = x, m/tsr = 1, pin = 1, flag0 = x;
ram p <- @SERVO_POSITION.

SERVO_RISING:
  chan pac := low;
  if hsq1 = 0 then goto SERVO_RISING1, flush. (*is channel reversed?*)

  au p := !p.      (*reverse position*)

SERVO_RISING1:
  au sr := p;
  ram diob <- @SERVO_DELTA.

  au a := 0;      (* clear a for multiply *)
  ram p <- @SERVO_MIN_COUNT.

  au dec := #7;  (* setup multiply *)
  chan clear flag0.

  repeat;      (* multiply & divide *)
  au a :=>> a + diob, shift.

  au sr := sr, ccl.      (*****)
                        (* round result? *)
  if n = 0 then goto SERVO_RISING2.  (*****)

  au p := a + p.      (*UNFLUSHED*)

  au ert := ert + p + 1;      (* round up *)
  chan write_mer,
    neg_mrl;
  end.

SERVO_RISING2:
  au ert := ert + p;
  chan write_mer,
    neg_mrl;
  end.

```

## DIFFERENCES BETWEEN THE TPU, TPU2, AND TPU3

The time processor unit (TPU), as any successful design, evolves and has been enhanced over the years. There are currently three versions of the TPU—TPU, TPU2, and TPU3—currently available on many Motorola microcontrollers.

The TPU was originally designed for the 68332 and has since been included on several 68300 and 68HC16 devices. Although the TPU proved extremely popular, the TPU2 was designed to meet the market's need for more program memory. The TPU3 was then derived to cope with the higher clock speeds required in the move to the PowerPC architecture. A summary of the hardware differences between the three TPU derivatives is shown in Table 1.

As shown in Table 1, the total accessible memory maps on the TPU, TPU2, and TPU3 are 2 KB, 8 KB, and 8 KB, respectively. The cost of moving to a larger memory map was the loss of the “no parameter preload” option. This was necessary as the entry addresses on the TPU2 and TPU3 required two extra bits. On the TPU2 and TPU3 a paged memory system is used with 2 KB of memory per page. The entry table can jump to a state within any of the 2-KB banks but, once a bank has been entered, it cannot jump to another bank. It is possible to have multiple entry tables allowing more than one set of TPU functions in ROM, RAM, or flash memory at once.

The parameter RAM on the TPU leaves the two most significant parameters on each channel, except channels 14 and 15, unimplemented. This saves space on the TPU but makes it difficult to write functions that use the parameters from more than one channel. This arrangement has been changed in the move to the TPU2 and TPU3 and extra parameters have been added to accommodate more complex functions.

Two new features on the TPU2 and TPU3 are the soft reset and output-disable features. The soft reset feature allows the TPU to be reset independently of the

microcontroller and the output disable feature allows the high-speed turning off of outputs. When the output-disable feature is enabled, there is only one gate delay between the signal being asserted and all of the TPU outputs going tristate. This is extremely important for applications like motor control, where it is essential that outputs be disabled as soon as possible after an error occurs.

As well as hardware changes between the TPU derivatives, microcode changes have also been implemented. A summary of the changes to the microcode is shown in Table 2.

All three TPU derivatives use a unique greater-than-or-equal-to comparator feature which allows the TPU to determine if a time is in the past or the future. This is an extremely important feature for real-time systems where signals must be generated as accurately as possible. When performing the greater-than-or-equal-to calculation, only 15 bits on each of the TPU's two 16-bit timebases are available because one bit is needed for the greater-than comparison. As some users indicated the need for the full 16-bit timebase range, the TPU2 and TPU3 were developed with an alternative equal-only feature. This feature can be used where the full range is required but the greater-than comparison is not necessary.

There are four other microcode changes between the TPU, TPU2, and TPU3:

- Flags per channel—when writing TPU microcode, software flags are useful because they are simple to set and test. As an enhancement to the TPU, a third software flag has been added to each channel on the TPU2 and TPU3.
- Current pin state—in the TPU, when testing the state of a pin, the state is latched on entry. Microcode instructions are required to obtain the current

	TPU	TPU2	TPU3
Memory map	2 KB	8 KB	8 KB
Internal memory	2 KB ROM	4 KB ROM	4 KB ROM
Emulation memory	2 KB SRAM	4 KB FLASH	6 KB SRAM
Parameter RAM	200 bytes	256 bytes	256 bytes
Input digital filter	4 clocks	2,4,8,16,32,64,128 or 256 clocks	2,4,8,16,32,64,128 or 256 clocks
T2CLK edge detect	Rising only	Rising, falling or both	Rising, falling or both
Soft reset	No Yes	Yes	Yes
Hardware output disable	No	Yes	Yes
Minimum TCR1 resolution	4 clocks (160 ns @ 25 MHz)	2 clocks (80 ns @ 25 MHz)	2 clocks (50 ns @ 40 MHz)
TCR1 prescaler divider	4 or 32	2, 4 or 32	Even values from 2 to 64
TCR2 prescaler divider	1, 2, 4 or 8	1, 2, 4 or 8	1, 2, 4, 8 or 16

Table 1—Here's a look at the features of the three TPU options.

state. On the TPU2 and TPU3, it is possible to directly test the latched and current pin states.

- Negate MRL/TDL—two new instruction formats that have been added are “negate match recognition latch” (MRL) and “transition detect latch” (TDL). These instructions clear a latch set after the occurrence of an event.
- RAM zero operation—on the TPU, the only way to clear a parameter location is to load zero into a register and subsequently store that register. With the RAM zero operation, it’s possible to clear a memory location using the RAM subinstruction. The RAM zero feature can also be used to clear the “P” and “DIOB” registers that are used to access parameter RAM.

	TPU	TPU2	TPU3
Timebase Match mode	Greater than or equal	Greater than or equal and equal only	Greater than or equal and equal only
Flags per channel	2	3	3
Current pin state condition	No	Yes	Yes
Negate MRL/TDL in format 3.	No	Yes	Yes
Clear DIOB, P or parameter with RAM sub-instruction	No	Yes	Yes

Table 2—Different features means that the microcode for each derivative will also be slightly different.

the TPU3 on the Motorola MPC555 but will work on any version of the TPU. The complete code for the servo function can be obtained from the Circuit Cellar website.

## DEMONSTRATION

To give a working example, I wrote a sample program for using this servo TPU function. The program, which

reads one input knob and controls two output servos, was written for the Motorola MPC555 Evaluation Board as shown in Photo 1. The program configures the TPU, inputs custom microcode, and sets up the TPU to run the servo function on two channels.

The queued analog-to-digital converter (QADC) is configured to con-

vert one channel continuously with the QADC value from the input knob and is then used to drive two servos in a continuous loop. One servo is run in normal mode while the other is run in reverse mode to complement each other as the knob is moved.

The MPC555 is a complex device with hundreds of registers. To make the device easier to use, there’s an MPC555 header file that defines all the registers and associated bit fields. The MPC555 also has a utilities header that provides macros to make using the on-chip TPU simpler.

In order to run the servo program, the user must first calculate the MIN\_COUNT and MAX\_COUNT servo values. Doing this depends on how the TPU is configured. When determining how fast to run the timebases, all functions running on every TPU channel must be taken into consideration. To get the best resolution on the output signal, the timebase should be run as fast as possible (constrained only by the fact that MAX\_COUNT must be less than \$8000 timer counts).

For this sample program, the MPC555 has been configured to run at 40 MHz with the timebase set to system clock divided by four, giving a resolution of 100 ns. For a typical servo with a MIN\_COUNT of 1 ms and MAX\_COUNT of 2 ms, MIN\_COUNT and MAX\_COUNT values of \$2710 and \$4E20 respectively will result. In addition to defining the MIN\_COUNT and MAX\_COUNT values, the frequency of repetition of the output signal must also be established. The REPEAT parameter has a resolution of \$8000 counts which, with a timebase of 100 ns, equates to 3.2 ms. As servos should be refreshed every 20–30 ms, and the formula for calculating the repeat value is [REPEAT × \$8000], setting REPEAT to seven gives a safe value of 22.6 ms. This is the value used in the given sample program.

The demonstration program for running the servo function can be found on the *Circuit Cellar* web site.

## YOU’RE IN CONTROL

From the information given, it is obvious that the TPU is an extremely

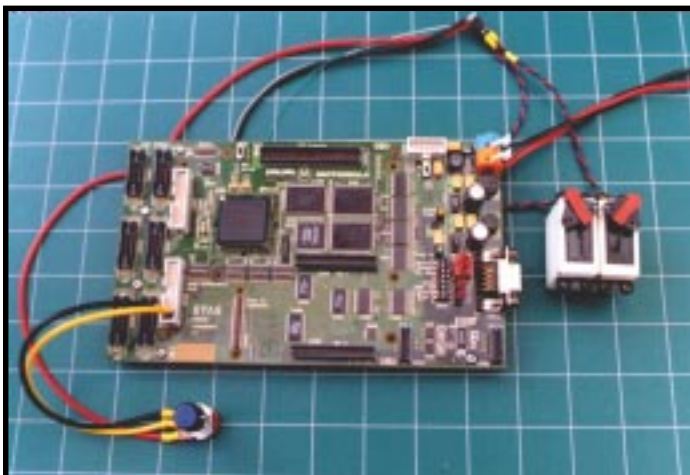


Photo 1—The MPC555 EVB runs demo code using a potentiometer to control two servos (one in normal mode and one in reverse mode).

```

State 0 : INIT      Select timebase and set-up pin.
                  Calculate DELTA = MAX - MIN.
                  Force immediate match to start generating control signal.
State 1 : UPDATE   If pin is high then
                  goto state 2.
                  Else
                  End
State 2 : RISING   Set-up pin for falling edge.
                  Clear flag 0.
                  Calculate falling edge time (POSITION * DELTA) / 256.
                  Round answer
                  Add to MIN_COUNT and ERT to get edge time.
State 3 : FALLING  Set-up next match.
                  If REPEAT = 0 then
                    set pin action
                  else
                    set flag 0 (do wait state next).
                  Set REPEAT_COUNT = REPEAT.

```

**Table 1**—Here's a detailed description of each state.

efficient way of controlling servos without causing any additional loading to the CPU. The entire servo function described here is implemented with only 23 TPU instructions and a single channel running the function only loads the TPU3 by less than 0.175% when running at 40 MHz. The minuscule loading requirement placed on the TPU leaves the door open for enhancements such as a 10-bit version of the function or an option for higher resolution. The future of the powerful TPU is limited only by the imagination of the programmer. ☐

*Jeff Loeliger is a senior staff engineer in the Advanced Vehicle Systems Division at Motorola SPS Europe. With over eleven years of experience in microprocessors, he has become a recognized expert on the Motorola TPU. You may reach him at [Jeff.Loeliger@motorola.com](mailto:Jeff.Loeliger@motorola.com).*

## SOURCES

MPC555  
 Motorola  
 (800) 521-6274  
 (512) 328-2268  
 Fax: (512) 891-4465  
[www.mot-sps.com](http://www.mot-sps.com)

## REFERENCES

- J. DiBartolomeo, "TPU: A Coprocessor for Timing Function", *Circuit Cellar* 102-105, 1999.
- Motorola, "MPC555 User's Manual", [www.mcu.motsps.com](http://www.mcu.motsps.com).
- Motorola, "Time Processor Unit Macro Assembler Reference Manual", TPUMASMREF/D2, 1994.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or [www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).

