design ideas

Edited by Bill Travis and Anne Watson Swager

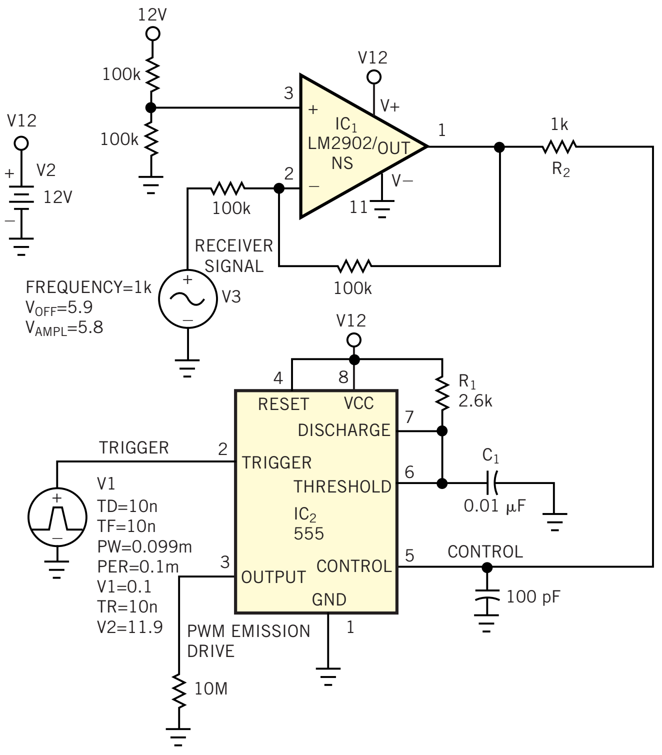
PWM circuit controls sensor's AGC

Dongjie Cheng, Allegro Microsystems, Willow Grove, PA

All electronic sensors have their limits on working distances and environmental tolerances. *Dynamic range* defines a sensor's maximum allowable variations in the signal amplitude. ACG (automatic gain control) finds widespread use in systems to extend the dynamic range. Applications using photoelectric or ultrasonic techniques involve both emission and detection energy. In many cases, the emission first establishes a background receiver signal as a reference, and the receiver monitors the signal and detects any changes against this reference. It is often desirable to maintain the background signal at a moderate level so that the sensor works within its limits. A signal that is too weak cannot produce a significant SNR, and a signal that is too strong can disable the sensor by saturation or produce overheating of the sensor. Designers use different techniques to achieve satisfactory signal handling. The simplest method might be adjusting emission power or the receiver's gain by manually configuring jumpers or switches. An example of an AGC solution is to let a µP constantly adjust emission to a suitable level. **Figure 1** shows a simple PWM-based technique for maintaining the reference signal at an ideal level.

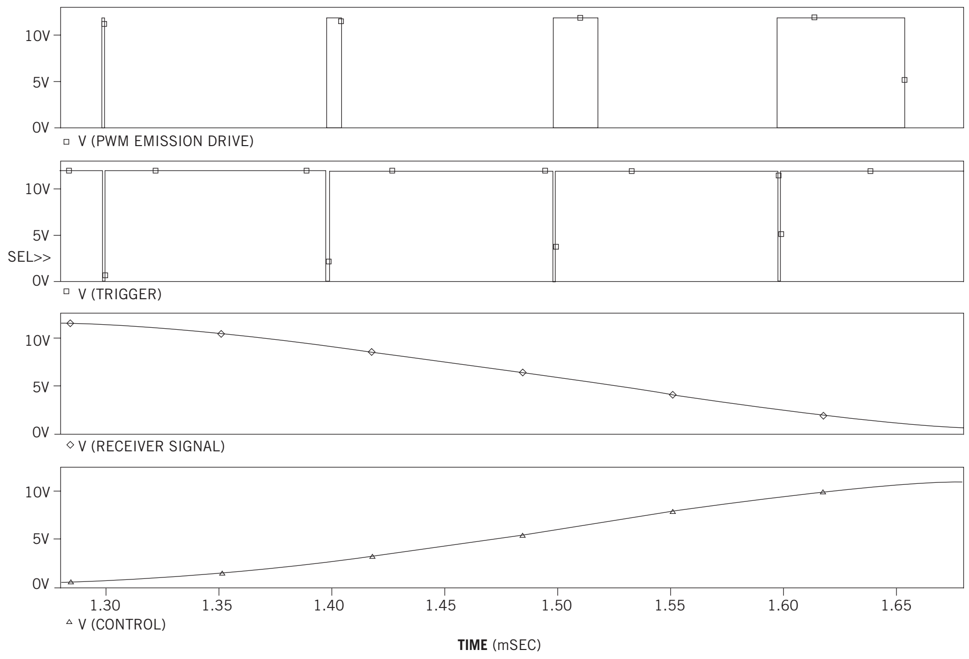
The concept embodied in **Figure 1**'s technique is that, if the sensor sees a strong background signal, the circuit slowly reduces the emission intensity at the fundamental frequency by reducing the pulse width of the emission drive signal. We verified the idea by using OrCAD/Cadence PSpice modeling, followed by experimental verification. The voltage source, V3, generates a 1-kHz sinusoidal signal named receiver signal. **Figure 2** plots this signal for a half-cycle in the third panel down. The model simulates the receiver's filtered, slowly varying background signal. IC_{2A} functions as an inverter. Its output passes through the lowpass filter R_2/C_2 to generate the Control signal in **Figure 2**, fourth panel down. IC_2, a 555 timer, acts as a pulse-width modulator. Its output is a pulse train named PWM emission drive (**Figure 2**, first panel). The frequency of the PWM emission drive follows the 10-kHz trigger signal, modeled with V1 and plotted in **Figure 2** in the second panel down. The modeled results in **Figure 2** indicate a negative correlation between the width of PWM emission drive and the level of receiver signal. As the receiver signal declines, the control signal widens the puls-

Figure 1



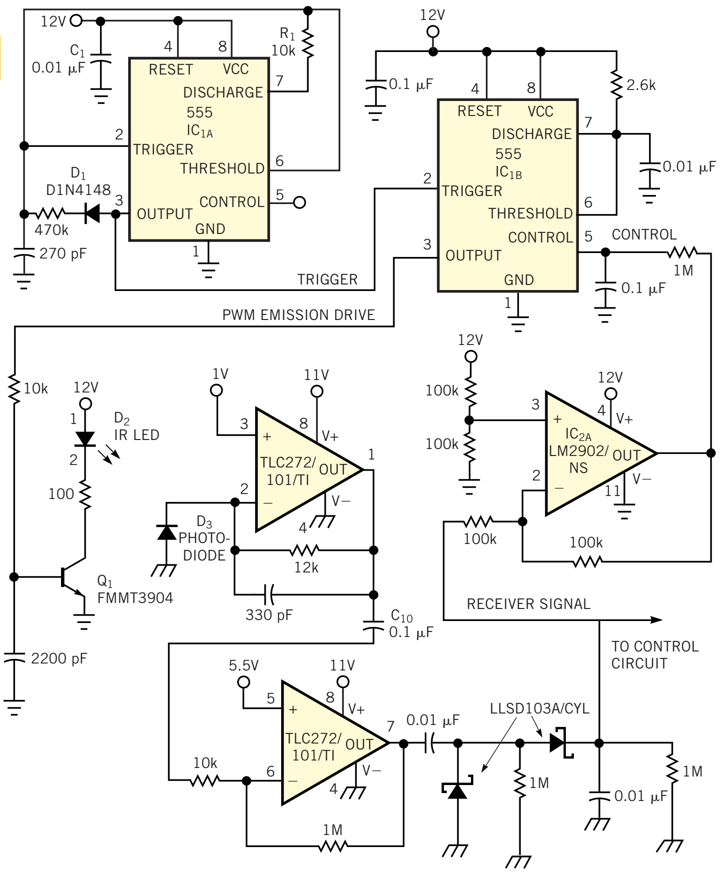
A PWM circuit provides an AGC function to increase a sensor's dynamic range.

Figure 2



The PWM emission-drive signal gains width (top panel) in response to a declining receiver signal (third panel down).

Figure 3



This circuit proves the validity of the PSpice model for the PWM AGC circuit.

es of PWM emission drive to boost the sensor's emission power at the fundamental frequency.

The process maintains the width of the PWM signal with the width fluctuating around a stable point. This point depends on the signal strength. If a sensor works near its upper range limit, the signal remains weak despite the increased pulse width, so the stable point shifts toward the maximum pulse width. In this Design Idea, the sensor is sensitive to the 10-kHz fundamental frequency because of a bandpass filter. Therefore, a 50% PWM duty cycle yields the maximum signal amplitude. On the other hand, if a sensor operates near its lower range limit, the pulse width converges to a very narrow width for a medium background signal. The time constant R_1C_1 in **Figure 1** also affects the location of the stable point. A longer time constant pushes the PWM stable point to a higher duty cycle and vice versa. In this case, the R_1C_1 time constant is half the trigger period (0.05 msec). You usually need to perform a test to determine R_1, C_1, or both. The test ensures that 50% is the maximum duty cycle from the modulator. The Trigger signal must be a narrow, negative pulse train.

Figure 3 shows a hardware setup to test the PSpice model, using an infrared sensor. IC_{1A} and IC_{1B} are 555 timers (the dual TLC556 or LM556). IC_{1A} is a free-running oscillator that supplies the 10-kHz Trigger pulses for the IC_{1B} PWM circuit. IC_{1B} drives the IR LED and Q_1. The 11, 1, and 5.5V voltages come from filtered voltage sources. A transimpedance amplifier first amplifies the 10-kHz photocurrent from the IR photodiode. The signal then goes through a second-stage amplifier and then undergoes filtering and peak detection to generate the quasi-dc receiver signal. IC_{2A} then inverts the signal to provide the PWM control. The PWM emission drive drives the IR LED in such a way that the IR photodiode receives medium-radiation intensities. Because the receiver's signal amplitude is proportional to the pulse width of PWM emission drive, the sensor constantly tries to reverse any trend of the background signal. In varying the distance between the IR LED and the photodiode, an oscilloscope showed signal behavior as predicted by the model.

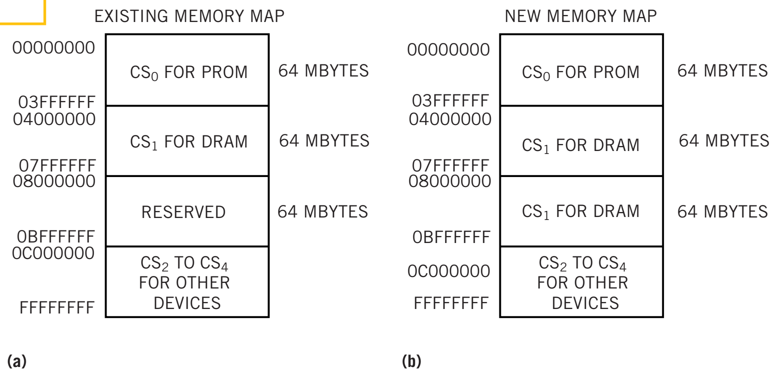
Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Two gates expand ASIC's memory-decoding range

Vinh Hoang, Ericsson Inc, Brea, CA

Many electronic circuits implement chip-select lines on an ASIC. From the beginning of the design cycle, the chip selects, CS_0 to CS_4, have defined bases on the memory map (**Figure 1**). Adding functions to the product requires increasing the DRAM space. Now, you must redesign the ASIC so the chip select, CS_1, can accommodate the new memory space of 04000000 to 0BFFFFFF.

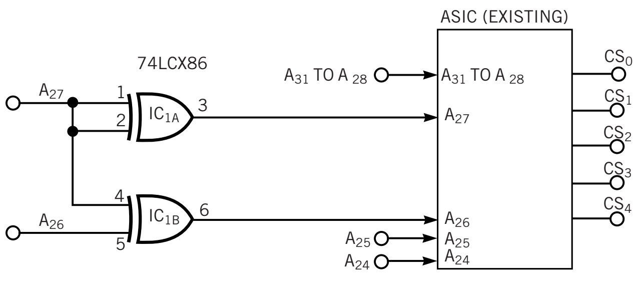
Figure 1



As you modify a product, you may have to expand the product's memory space (a) to accommodate more DRAM (b).

The circuit in **Figure 2** uses two external exclusive-OR gates to expand memory-address-decoding space for the CS_1 signal from the initial range of 04000000 to 07FFFFFF to 04000000 to 0BFFFFFF. When address line A_{27} is low, exclusive-OR gates IC_{1A} and IC_{1B} allow A_{27} and A_{26} signals to pass through unchanged. In this case, the ASIC decodes the address range 04000000 to 07FFFFFF, according to the existing memory map. However, when address line A_{27} is high, which accesses the address range 08000000 to 0BFFFFFF, both IC_{1A} and IC_{1B} act as inverters. Thus, the circuit inverts lines A_{26} and A_{27}, so that the ASIC now sees the address range 04000000 to 07FFFFFF instead of 08000000 to 0BFFFFFF. The function of the exclusive-OR gates is to map the CS_1 selected address range of 08000000 to 0BFFFFFF to 04000000 to 07FFFFFF.

Figure 2



Two exclusive-OR gates expand the memory-address-decoding space for CS_1.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Dual dc-motor-speed controllers navigate robots

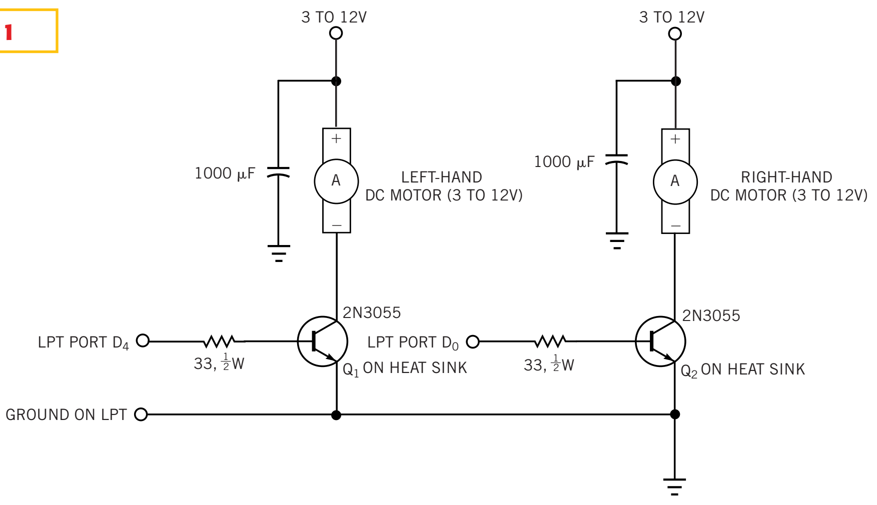
Shyam Sunder Tiwari, Robotics Software Pvt Ltd, Gwalior, India

Mobile robots need simple and lightweight dc-motor speed-control hardware that can work with low-power batteries. Flip-flop type navigation systems allow only one motor to operate at a time while the other motor remains off. Navigation direction changes alternatively even when the robot has to navigate through a straight path. Line-tracker robots are of this type.

Zigzag motion lets the robot look at the track toward its left and right such that it can correct its path if necessary. You can drive both motors if this type of motion is undesirable. The circuit in **Figure 1** has two independent motor-speed-control channels: one for a righthand-side motor and the other for a lefthand-side motor. Power to each motor is pulse-width-modulated using a Basic computer program (**Listing 1**). The power-driver circuit uses npn power transistors, Q_1 and Q_2. These transistors have high-power-kicking ability that the robotics require. The PC's parallel port directly controls the base of these transistors. LPT port data bit D_0 operates a righthand-side motor, and data bit D_4 operates a lefthand-side motor. Level one at the port pin turns on the motor power, and level zero turns off the motor power. If both D_0 and D_4 are set to one, then both motors operate together. Reverse control does not occur. Thus, only one motor needs to operate to turn the robot backward until rotation is complete. You can add feedback sensors to the hardware. These sensors are necessary to know the position of the robot. The circuit works for small dc motors operating from a power source in the range of 3 to 12V.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Figure 1



Independent control channels drive righthand and lefthand dc motors.

LISTING 1–MOTOR-SPEED CONTROL

```
        CLS
        REM LPT1 port is located at address decimal 888
        REM
start:  INPUT "motor under action (1= left 2=right 3=both) =", m
        IF m > 3 GOTO start
        IF m < 1 GOTO start
        REM
        REM select the motor to be operated
        IF m = 1 THEN w = &H0F0 : REM left motor
        IF m = 2 THEN w = &H0F : REM right motor
        IF m = 3 THEN w = &H0FF : REM both motors
        REM
        REM Get the power factor for speed control
        REM
power:  INPUT "motor speed power factor (scale 10 to 80) =", p
        IF p > 80 GOTO power : REM check for upper limit
        IF p < 10 GOTO power : REM check for lower limit
period: INPUT "motor power on time (range 1-1000 seconds) =", s
        IF s > 1000 GOTO period : REM maximum time in seconds
        IF s < 1 GOTO period
        REM
        REM compute the PWM parameters here
        REM
        H = 20 * p
        REM H is high level PWM control output
        L = 20 * (80 - p)
        REM L is low level PWM control output
        CLS
        LOCATE 10, 5
        PRINT "
        LOCATE 11, 1
        FOR i = 1 TO 8
```

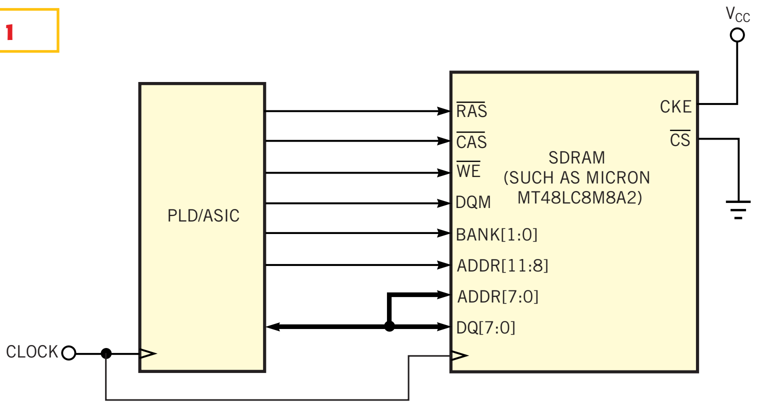
```
        PRINT ":";
        FOR j = 1 TO 9
        PRINT ".";
        NEXT
        NEXT
        LOCATE 10, 1
        FOR i = 1 TO p
        PRINT "-";
        NEXT
        PRINT ">": k = 0
        ON TIMER(1) GOSUB time
        TIMER ON
        REM

repeat: IF k > s GOTO done
        GOSUB control: REM Endless loop
        GOTO repeat
done:   STOP
time:   LOCATE 12, 25
        k = k + 1
        PRINT "(power="; p; " time="; k; "seconds)"
        RETURN
        REM Motor power switching subroutine
control: FOR i = 1 TO H
        OUT 888, w: REM 888 is the address of LPT1 data port
        NEXT i
        FOR i = 1 TO L
        OUT 888, 0
        NEXT i
        RETURN
```

SDRAM interface slashes pin count

Tim Hellman, M&M Consulting, Concord, MA

Many designs need deep buffering but don't require ultra-high-memory bandwidth. Examples include image and audio processing, as well as some deep-FIFO applications. These designs often use a single ×8 SDRAM device that connects to an FPGA or ASIC. This approach solves the buffering problem but also burns a lot of valuable pins, which can be as many as 27 for a single SDRAM device. The design in **Figure 1** takes advantage of the burst counter inside the SDRAM to reduce this pin count to 18 by multiplexing the lower eight address lines with the data. The efficiency loss is low; the design requires only one extra clock during the write burst. **Figure 1** uses an 8Mx8, 125-MHz SDRAM, but this technique works with any SDRAM.

Figure 1



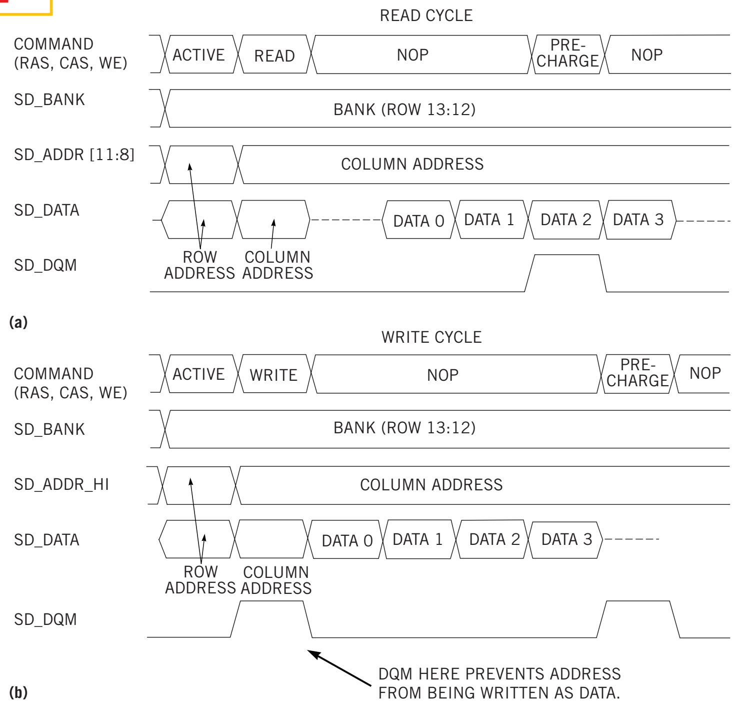
To reduce the interface-pin count to 18, you can take advantage of the burst counter inside the SDRAM by multiplexing the lower address and data lines.

The read- and write-cycle timing diagrams reveal the secret (**Figure 2**). The **figure** shows a burst of 4, but any power-of-2 burst works. These diagrams assume a 50-MHz system clock, a read latency of 2, and a full-page burst. During the read cycle (**Figure 2a**), the data bus is inactive during the initial portion of the cycle (when the row and column addresses are presented), so there's no problem with using the data bus to carry address data. A precharge command ends the burst and prepares the RAM for the next access.

For the write cycle, however, some trickery is necessary (**Figure 2b**). Normally, the first byte of write data is presented to the SDRAM with the Write command, along with the starting column address for that burst. By asserting the DQM (data-mask) signal, the SDRAM ignores the data lines during this phase, thus allowing them to be used for the column address.

Note that the DQM signal does not prevent the internal column address from incrementing, however. Thus, the write-column address presented with the Write command must be one less than the desired burst starting address. For FIFO designs, this requirement is trivial because you can initialize the write-address col-

Figure 2



During the read cycle (a), the data bus is inactive during the initial portion of the cycle so the data bus can carry address data. During the write cycle (b), asserting the DQM command causes the SDRAM to ignore the data line during this phase, which allows the data lines to carry the column address.

umn counter to −1 rather than 0. The column-address counter in the SDRAM wraps around at the end of the column, so this approach works even at the beginning of a column.

You can download a simplified version of a FIFO controller that uses this technique, described in the Verilog language, from *EDN*'s web site, www.ednmag.com. Click on "Search Databases" and then enter the Software Center to download the file for Design Idea #2659. The listing omits some of the details, such as SDRAM refresh/init cycles, and FIFO flags, to highlight the portions relevant to this design. This controller uses a simple eight-state finite-state machine to generate the SDRAM control signals and uses a pair of row/column counters to keep track of the FIFO put/get pointers. The special initialization and incrementing of the write row and column pointers satisfies the requirement that the write column start off one behind the desired write address. The code occupies 35% of a small Xilinx SpartanXL-S10 device, and runs at 50 MHz. For the sake of example, all of the outputs are combinatorial, but a true high-speed design should use registered I/O.

You can extend this idea to ×16 SDRAMs and to multiplex a few more of the address lines while getting a boost in memory bandwidth. If you do extend the idea, be careful with SDRAM line A_{10} because this line has special meaning during some SDRAM commands. You can also use this technique with double-data-rate SDRAMs.

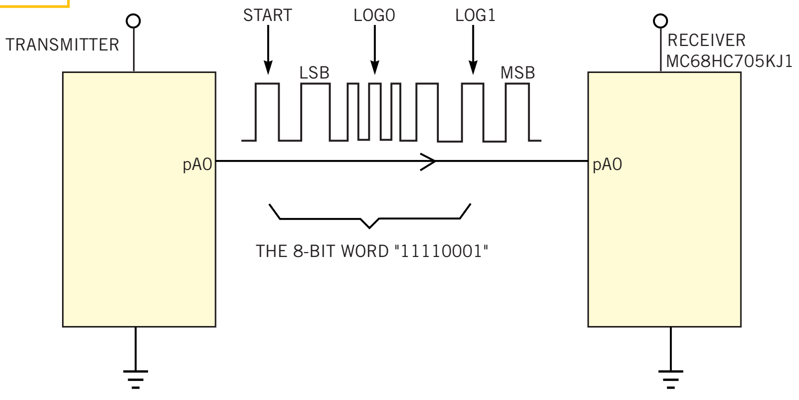
Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Minimize communication time between small µCs

Abel Raynus, Armatron International Inc, Melrose, MA

When designing a multicontroller system, it is convenient to organize the communication between µCs via one wire line. Unfortunately, low-end µCs have no serial-interface capabilities like their more expensive counterparts. Low-end µCs have no SCI (serial-communication-interface, SPI (serial-peripheral interface), or SIOP (simple serial-I/O port). Thus, a designer needs to use software tools to create a serial interface (**Reference 1**). One approach, which uses the external interrupt for message receiving, results in message duration of 13.5 to 21.5 msec. For many applications, this time is unimportant. However, some applications may require you to minimize the message time as much as possible, especially when both the timer and external interrupts are in use. The µC cannot simultaneously execute these interrupt requests, and the external interrupt has a priority. Hence, a message and, therefore, the external-interrupt-service routine that are too long can affect the program-process timing related to timer interrupt.

Figure 1



The data word from the transmitting µC connects to Pin pA0 of the receiving µC, which you program as an external-interrupt input.

You cannot minimize the message time by just changing some number in the program. You must instead use a time-measurement concept that leads to modification of the whole program (**Figure 1**). The data word from one µC to another comes to Pin pA0, which you program as an external-interrupt input. When an external interrupt occurs, the µC can process this interrupt only when the current instruction execution is complete. This waiting time is always unpredictable and can range from zero to the time of the longest instruction in the program (**Figure 2**). So, this time differs for each user program. Nevertheless, if a program has no exotic instructions, such as SWI (software interrupt) and MUL (multiplication), then the longest instruction can execute in six cycles, which takes 3 μsec for an oscillation frequency of 4 MHz (**Reference 2**). Saving the contents of the CPU registers on the stack and loading the program counter with an external-interrupt vector address take nine more cycles.

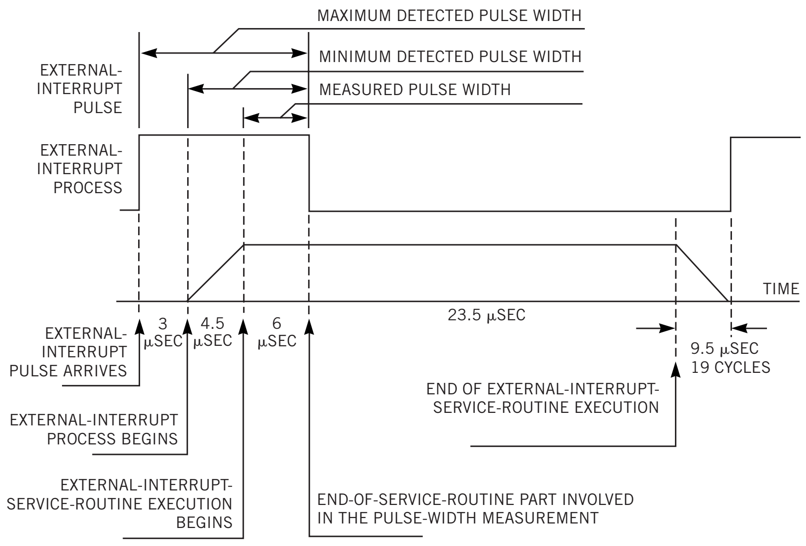
Only at this point can the interrupt-service routine begin the pulse-width measurement. (You can download the message receiver and transmitter programs *EDN*'s Web site, www.ednmag.com. Click on "Search Databases" and then enter the Software Center to download the file for Design Idea #2682.) After executing three instructions (lines 36 to 38 of the message-receiver program), this process completes in 12 cycles, or 6 μsec. Hence, the width of an external interrupt pulse cannot be less than 10.5 μsec.

From another side, it is more convenient for the transmitter part of the program to use only one subroutine, called Pulse, to generate all the pulses with different widths by changing the number of the loop repetitions (X) only. The loop duration is 5 μsec (lines 43 to 46 of the message-receiver program), so the generated pulse is equal to 5X+2.5 μsec. You can choose the pulse-width values according to your project objectives. In this case, for LOG0 X=2, width=12.5 μsec; for START X=4, width=22.5 μsec; and for LOG1 X=6, width=32.5 μsec. However, as you can see from **Figure 2**, the pulses that the receiver measures are shorter by 4.5 μsec than transmitted pulses. So, the measured pulses have widths of 8, 18, and 28 μsec. The μC's internal free-running timer/counter is the clock. You can at any time read the value of the first eight stages of this counter from the (TCR) time-counter register. One time count is equal to four machine cycles, or 2 μsec. Note that the pulse-width measurement must be complete before the TCR overflow that happens every 2×256, or 512 μsec. In the program, the unit of pulse-width value calculated and put into register pW is one timer count (clk). So the widths of LOG0, START, and LOG1 pulses are 4, 9, and 14 clk, respectively. Remember that the moment of beginning and external interrupt processing could be at any time in the range of 3 μsec, or 2 clk, which causes the same variations in the measured pulse width. To overcome this problem, you select the proper pulse width within a fork, which should greater than or equal to 2 clk. In this case, the fork is equal to 4 clk. Finally, you use the following pulse selection logic: 3 clk< LOG0<7 clk, 8 clk<START<12 clk, and 13 clk<LOG1<17 clk.

The minimal interval between the two consecutive external-interrupt pulses should be more than the number of interrupt-service-routine cycles plus 19 cycles (**Reference 2**). The service routine's longest path occurs when the program is processing the widest incoming pulse (LOG1), which takes 47 cycles. Thus, the interval between pulses should be more than 47+19=66 cycles, or 33 μsec. Choosing the number of time-loop repetitions of X=10 results in a 52.5-μsec pause between pulses (lines 48 to 52 of the message-receiver program). With all the above chosen values, the longest message, $\$ff_{HEX}$. lasts 0.84 msec.

The design in **Figure 1** uses a one-time-programmable Motorola MC68-HC705JK1 (Motorola μC, but this idea is applicable to any μC. You would need to recalculate the timed values according to the μC's technical data.

Figure 2



MAXIMUM DETECTED PULSE WIDTH
MINIMUM DETECTED PULSE WIDTH
MEASURED PULSE WIDTH
EXTERNAL-INTERRUPT PULSE
EXTERNAL-INTERRUPT PROCESS
TIME
23.5 μSEC
3 μSEC
4.5 μSEC
6 μSEC
EXTERNAL-INTERRUPT PULSE ARRIVES
9.5 μSEC 19 CYCLES
END OF EXTERNAL-INTERRUPT-SERVICE-ROUTINE EXECUTION
EXTERNAL-INTERRUPT PROCESS BEGINS
EXTERNAL-INTERRUPT-SERVICE-ROUTINE EXECUTION BEGINS
END-OF-SERVICE-ROUTINE PART INVOLVED IN THE PULSE-WIDTH MEASUREMENT

The μC can service an external interrupt only when it finishes executing the current instruction, and the waiting time can range from zero to the time of the longest instruction in the program.

References

1. Raynus, Abel, "Single wire connects microcontrollers," *EDN*, Oct 22, 1998, pg 102.

2. MC68HC705KJ1/DTechnical Data, Revision 1.0, Motorola.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.