

# A 32 Channel Digital R/C Servo Controller

## *ABSTRACT*

Project #A3722



**H**ave you been longing to build that cool multi-legged walking robot with over 30+ R/C servos but haven't found a single controller that will allow you to drive them all at once? Well look no further! This project will show you how to build a 32 channel digital R/C servo controller to help bring your sophisticated robotic creations to life!

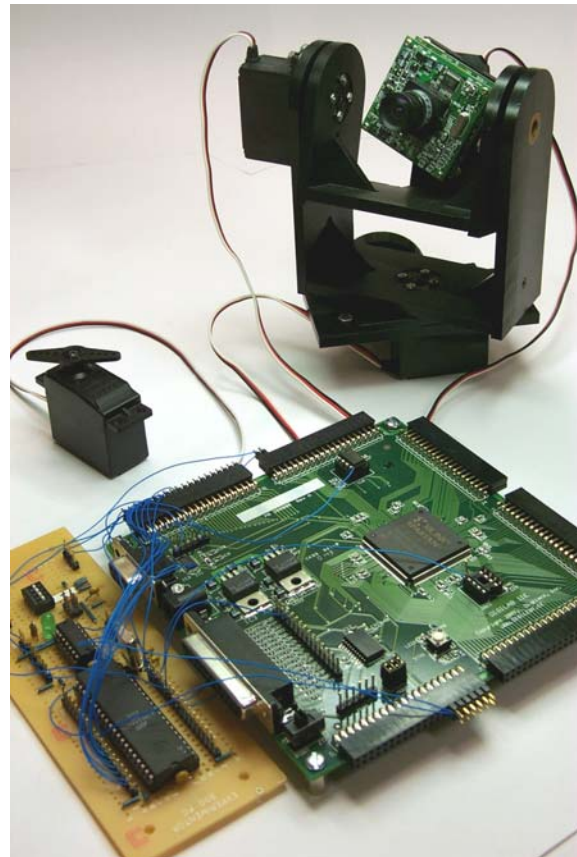
Radio Controlled (R/C) servos have enjoyed a big comeback in recent years due to their adoption by a new generation of robotics enthusiasts. Driving these versatile servos requires the generation of a potentially large number of stable pulse width modulated (PWM) control signals, which can be a daunting task. A simple solution to this problem is to use a dedicated serial servo controller board which handles all the details of the multi-channel PWM signal generation while being controlled through simple commands issued on a standard serial UART.

Most of the current serial servo controllers on the market offer a limited number of channels (1-16). They often also have a limited effective resolution of 8 bits or less, whereas the newer digital R/C servos provide up to 10 bits of resolution. Most existing designs rely on a microcontroller with firmware bit-banging used for PWM generation. Depending on the implementation, some controllers may exhibit PWM jitter, timing errors, or lack of PWM synchronization between channels which results in servo tracking delays.

The design introduced in this project departs from the traditional MCU-centric approach and instead combines the brute force of an FPGA and the higher level intelligence of the MCU to achieve some impressive specifications.

An array of 32 parallel channels of 16-bit accuracy, 12-bit resolution PWM generation units were implemented inside an FPGA. An Atmel ATmega8515L was used at the heart of the system. It's external memory bus was put to good use in interfacing with the memory-mapped array of 64 PWM registers (i.e. 32 x 16 bits) inside the FPGA. The many roles of the MCU include initializing the FPGA registers with user-configurable servo startup positions stored in the 8515L's internal EEPROM.

Also, the 8515L runs the serial communications protocol (compatible with Mini SSC-II protocol) and updates servo positions accordingly. In addition, the 8515L performs real-time velocity control computations on all channels with configurable velocity settings. Finally, the 8515L also checks all outgoing servo commands against user-defined min/max position values. In response to an external interrupt occurring at every PWM cycle, all current servo position values are refreshed in the FPGA's memory mapped PWM registers. This is done as a simple memory-to-memory transfer.



*Figure 1: Picture of the 32 channel digital R/C servo controller circuit board prototype driving a multi-axis robotic camera pan-tilt unit.*

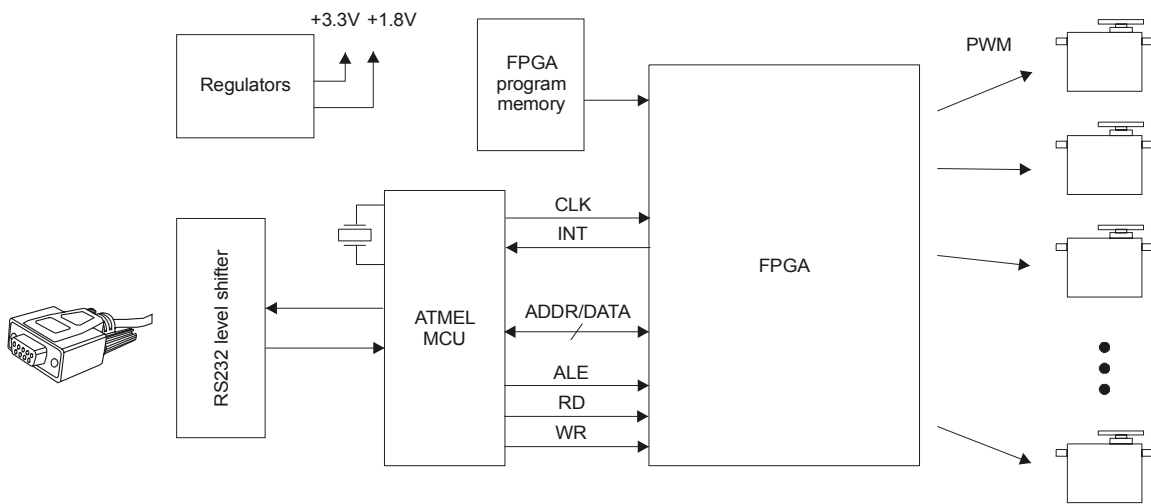


Figure 2: Overall serial servo controller circuit block diagram.

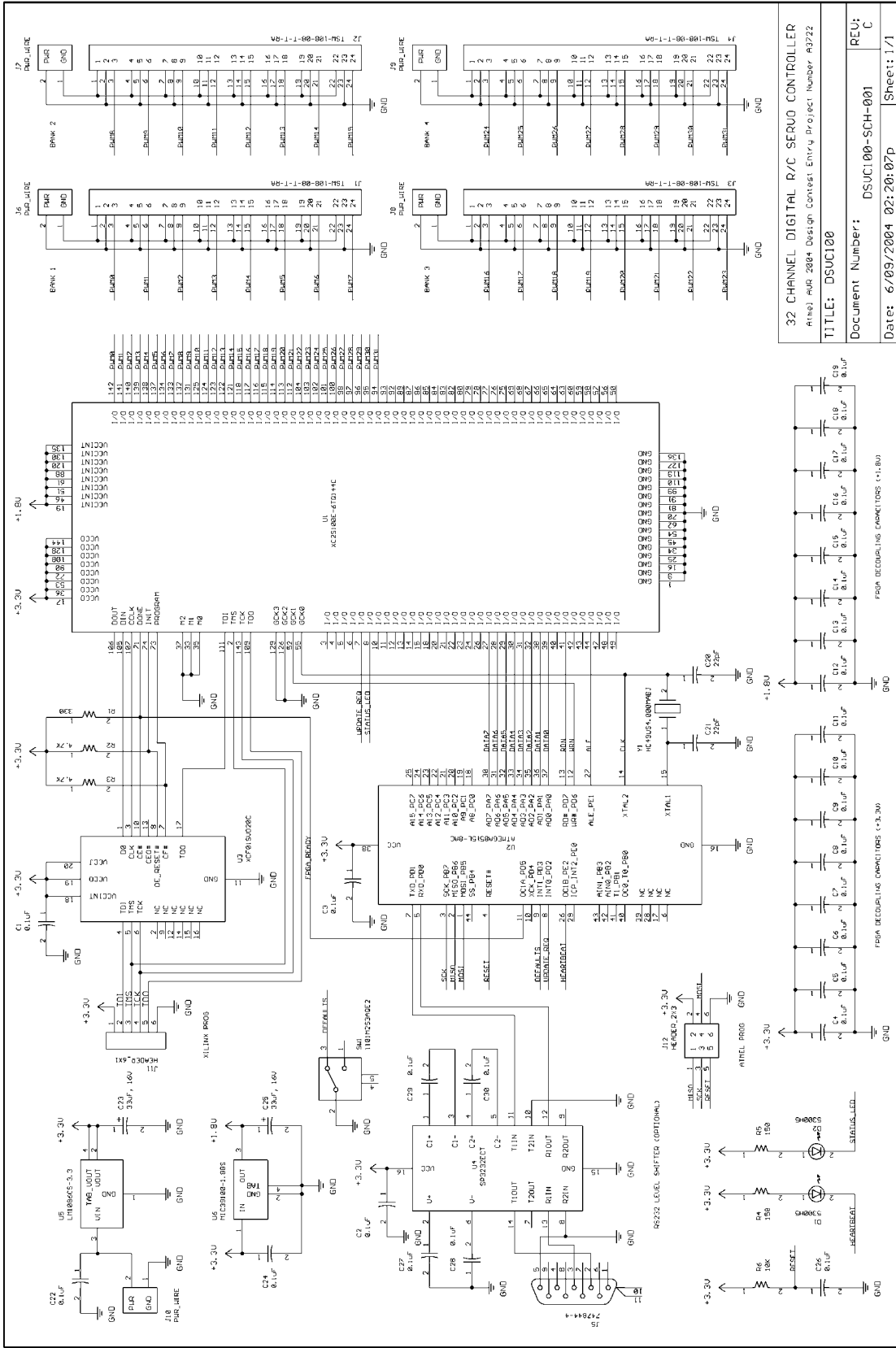


Figure 3: Full circuit schematics of the 32 channel digital serial servo controller.

32 CHANNEL DIGITAL R/C SERVO CONTROLLER  
 #ref: AVR 2004 Design Contest Entry Project Number A3722  
 TITLE: DSUC100  
 Document Number: DSUC100-SCH-001  
 Date: 6/09/2004 02:20:07P  
 REV: C  
 Sheet: 1/1

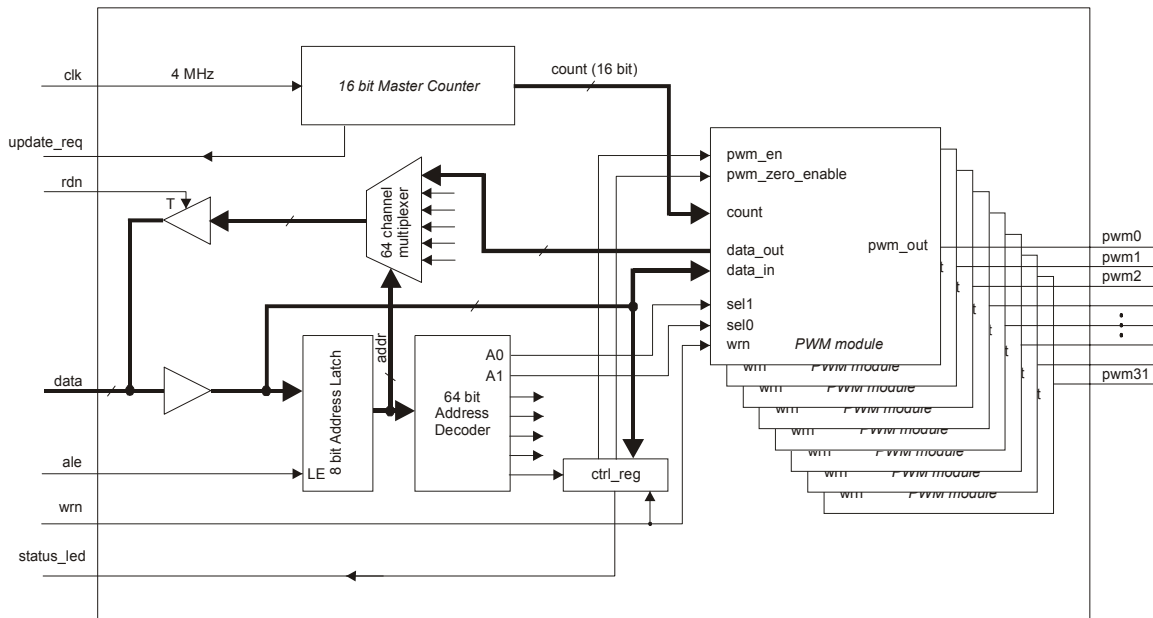


Figure 3: Internal architecture of FPGA.

```

--(vhdl code snippet)
.
.
.
process(wrn, sel0, sel1, data_in)
begin
    if (wrn'event and wrn='1') then
        if (sel1 = '1') then
            pwm_val(11 downto 4) <= data_in;
        elsif (sel0 = '1') then
            pwm_val( 3 downto 0) <= data_in(7 downto 4);
        end if;
    end if;
end process;

data_out <= pwm_val(11 downto 4) when (sel1='1') else
    (pwm_val( 3 downto 0) & "0000");

pwm_out <= '0' when (pwm_en = '0') else
    '0' when (pwm_val = "0000000000000000" and pwm_zero_enable = '1') else
    '1' when (count > "1111100000110000") else
    '0' when (count > ("000" & pwm_val(11 downto 0) & "0")) else
    '1';
    -- (shift left by one for 180 deg servo control mode (deltaT = 2msec)

```

Listing 1: FPGA code snippet (VHDL).

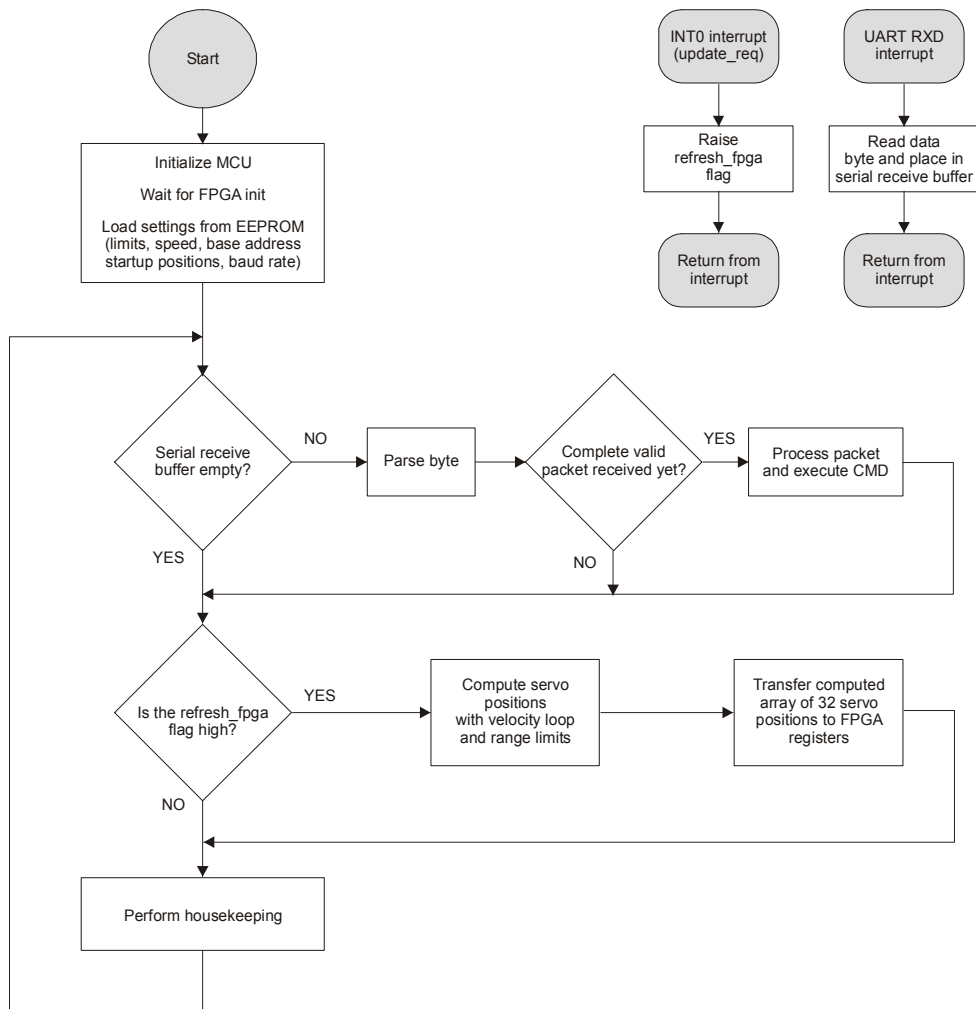


Figure 4: MCU firmware flowchart.

```

//-----
// Function name: main()
//
// Purpose: Main thread of execution
//
// Notes:
//-----
void main(void) // Main thread
{
    BYTE char_in; // Temp variable

    initialize(); // Initialize MCU
                 // Interrupts are now ON
    for (;;) // Infinite loop
    {
        // 1) Process serial stream commands timeslice

        if (!serial_rx_empty()) // Check if RX chars
        { // Read character
            char_in = serial_getchar();
            // Process serial stream
            command_process(char_in);
        }

        // 2) Update FPGA timeslice

        if (fpga_update_req) // Time to refresh pwm vals?
        {
            fpga_update(); // Yes, so update values
            fpga_update_req = FALSE; // Clear flag
        }

        // 3) Heartbeat LED update timeslice

        heartbeat_update(); // Heartbeat LED handler

        // 4) Reset watchdog timeslice

        WDR(); // Kick watchdog
    }
}

```

*Listing 2: MCU firmware code snippet.*

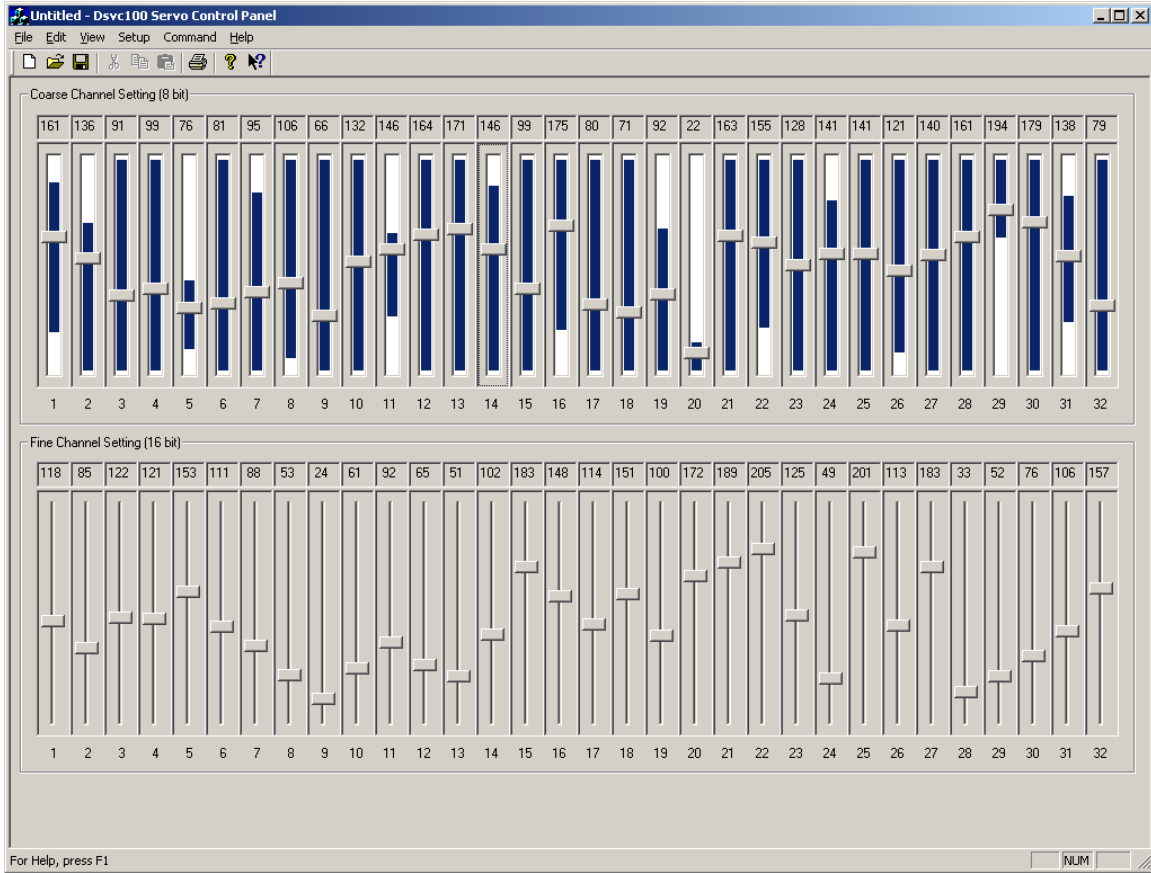


Figure 5: PC control software screen capture.