

Yaesu G-5500 Azimuth-Elevation Rotator Controller

Project eZ2966

Abstract

Rotator controllers are easy to buy or build. Most use either serial RS-232 or parallel port interfaces. While one or two use USB the USB is really just a USB to serial interface. My version of the Yaesu G-5500 Azimuth-Elevation Rotator Controller, based on the eZ80F91 Mini Enet Module, provides a basic web page interface, a telnet command interface and a serial command interface. (The serial command interface is intended to be used by the many Rotator Control Programs that are already available.) The system receives azimuth and elevation positions from the rotator using an analog-to-digital converter. Four output bits control the rotator (left, right, up and down). The four output bits and the position values connect to the G-5500 rotator control box which comes with the rotator. With some minor hardware and/or software changes, additional rotators can be controlled as well as different kinds of rotators.

Sample Code (rotator controller task)

```
static PROCESS Rotator_Task(void)
{
    unsigned i;
    Rotator_Command_Type command;
    DWORD current_time;

    while (1) {
        /* Run five times per second. */
        (void) KE_TaskSleep10(2);

        /* Read ADC Channel Values. */
        for (i = 0; i < DIM_Of(ADC_Channel_Table); i++) {
            /* Get one 10-bit converted ADC value. */
            ADC_Read(&ADC_Channel_Table[i]);
        }

        (void) KE_TaskGetTime(&current_time);

        /* Process rotator commands. */
        for (i = Rotator_Type_First; i <= Rotator_Type_Last; i++) {
            Rotator_Table_Type *rotator = &Rotator_Table[i];

            /* Get current position from ADC. */
            rotator->position =
                ADC_Channel_Table[rotator->adc_channel].ADC_Value;

#ifdef INCLUDE_COOLDOWN
            if (rotator->state == RC_Off) {
                /* Rotator motor is off, cooldown. */
                if (rotator->cooldown_count > 0)
                    rotator->cooldown_count--;
            }
#endif
        }
    }
}
```

```

} else {
    /* Rotator motor is on, need 3x cooldown. */
    rotator->cooldown_count += 3;
}
if (rotator->cooldown_count > (5 * 15 * 60)) {
    /* The rotator motor is rated for 5 minutes intermittent
    ** duty, let rest for 15 minutes. */
    RC_Set(i, RC_Off, "Cooldown");
    RC_Output_States();
    continue;
}
#endif

if (rotator->state != RC_Off) {
    /* Stall management. */
    if (rotator->stall_interval > 0) {
        /* It is not time to do a stall check. */
        rotator->stall_interval--;
    } else {
        /* Expired timer, time to do a stall check. */
        if(Near(rotator->position, rotator->stall_position)){
            /* The rotator is not moving,
            ** stop it just in case. */
            RC_Set(i, RC_Off, "Stall");
            RC_Output_States();
            continue;
        }

        /* Restart the stall interval timer. */
        rotator->stall_interval = STALL_INTERVAL;
        rotator->stall_position = rotator->position;
    }

    if (rotator->stop_time > 0 &&
        current_time > rotator->stop_time)
    {
        /* The duration period has expired.
        ** Turn the rotator off. */
        RC_Set(i, RC_Off, "Duration");
        RC_Output_States();
        continue;
    }

    if (rotator->stop_position >= 0) {
        /* Position control is enabled,
        ** and not already stopped. */

        /* Note, RC_Left == RC_Up and RC_Right == RC_Down
        ** so need i == ... */
        if (Near(rotator->position, rotator->stop_position)){
            RC_Set(i, RC_Off, "Position");
            RC_Output_States();
            continue;
        } else if (i == Elevation_Rotator &&
            rotator->state == RC_Up)
        {
            if (rotator->position > rotator->stop_position)

```

```

        RC_Set(i, RC_Down, NULL);
    } else if (i == Elevation_Rotator &&
               rotator->state == RC_Down)
    {
        if (rotator->position < rotator->stop_position)
            RC_Set(i, RC_Up, NULL);
    } else if (i == Azimuth_Rotator &&
               rotator->state == RC_Left)
    {
        if (rotator->position < rotator->stop_position)
            RC_Set(i, RC_Right, NULL);
    } else if (i == Azimuth_Rotator &&
               rotator->state == RC_Right)
    {
        if (rotator->position > rotator->stop_position)
            RC_Set(i, RC_Left, NULL);
    } else {
        /* None of the above, ignore. */
    }
    }
}

if (rotator->done)
    continue;

switch (command = rotator->command) {
    case Rotator_No_Change:
        break;
    case Rotator_Stop:
        RC_Set(i, RC_Off, NULL);
        break;
    case Rotator_Left: /* case Rotator_Up: */
        RC_Set(i, RC_Left_Up, NULL);
        break;
    case Rotator_Right: /* case Rotator_Down: */
        RC_Set(i, RC_Right_Down, NULL);
        break;
    default:
        // Should never happen.
        // Set to "XX Unknown".
        RC_Set(i, RC_Unknown, NULL);
        break;
}
}

/* Write the rotator states (off, left, etc.)
** to the output port(s). */
RC_Output_States();

for (i = Rotator_Type_First; i <= Rotator_Type_Last; i++)
    Status_Update(i);
}

void Rotator_Init()
{
    KE_TASK *t;

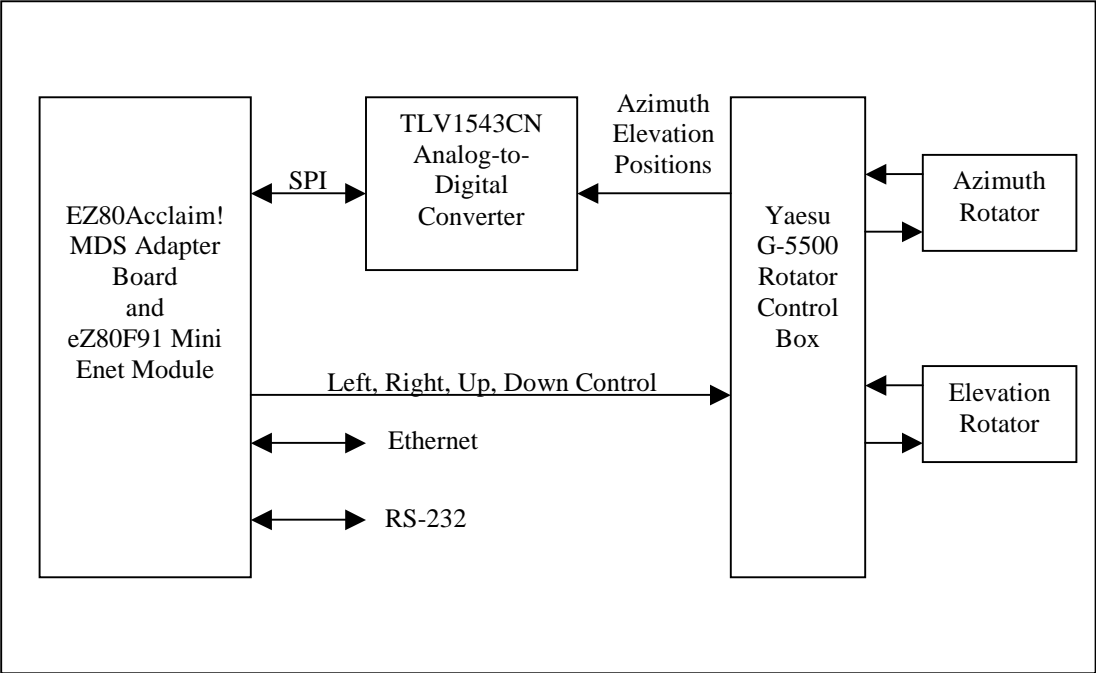
```

```
// Turn off the rotators.
RC_Set(Elevation_Rotator, RC_Off, NULL);
RC_Set(Azimuth_Rotator, RC_Off, NULL);
RC_Output_States();

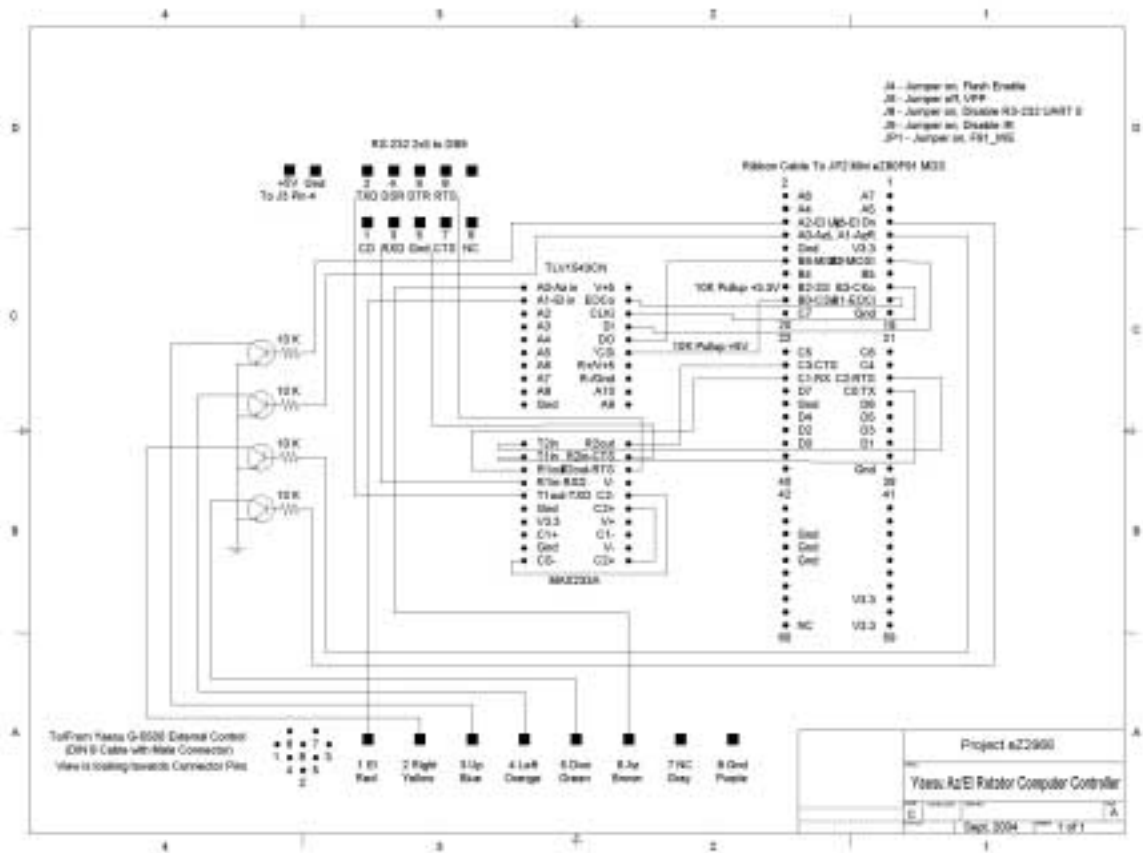
// 500 KHz Clock
// SPI_BGR = 50 MHz / (2 * Clock_Rate)
SPI_BRG_H = 0;
SPI_BRG_L = 50;

// Disable Interrupt; SPI Enabled; Master Enabled;
// Master idles in zero/low state;
SPI_CTL = 0x30;

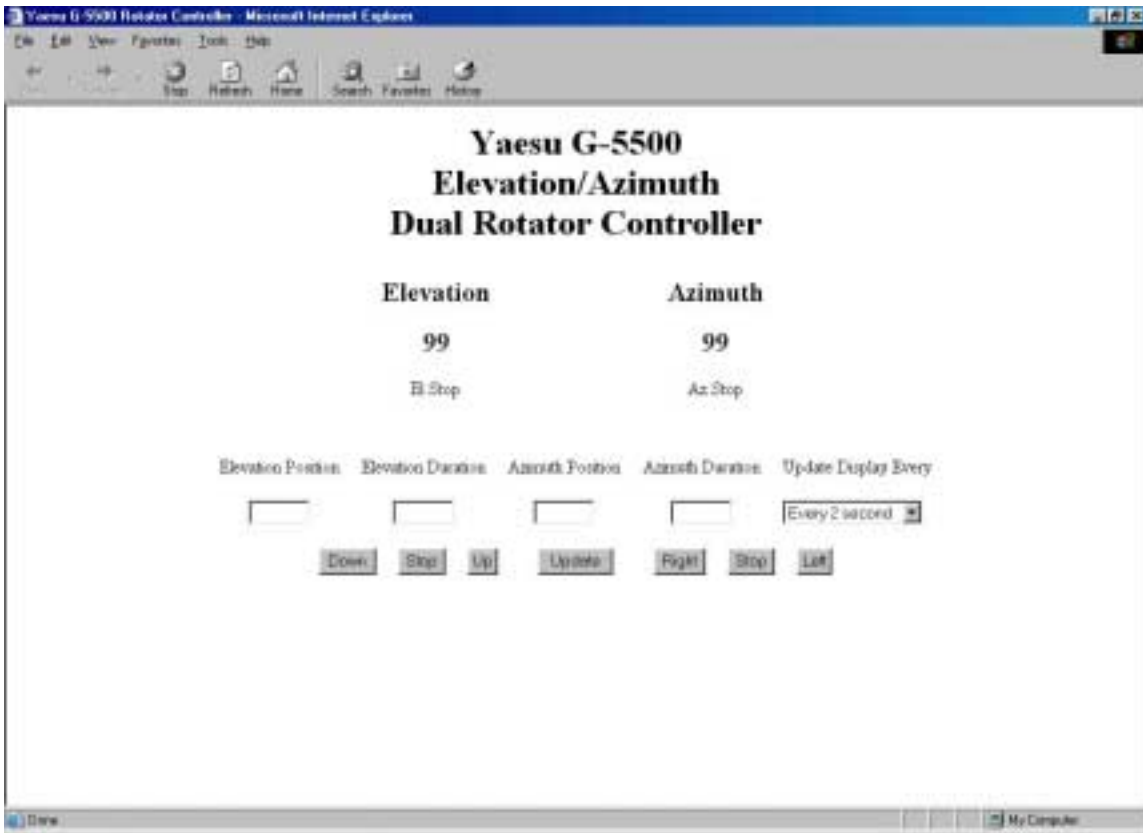
t = KE_TaskCreate(Rotator_Task, 2048, 10, "RotatorCntl", 0);
(void) KE_TaskResume(t);
}
```

Block Diagram



Schematic



Main Web Page