Running Motion (Positioning) commands from PLC

Option 1

Motion Controller and PLC running on myCNC controllers as independent tasks in Real Time multi-tasking environment. There is API to run motion commands from PLC. This features used for wide range of procedures like probing, homing, measure procedures etc.

Procedure **g0moveA** is used to send motion command from PLC to motion controller. This procedure has 3 parameters -

```
g0moveA(flags, axes_mask, distance);
```

- **flags**
  - bit 0 - absolute programming (0 - incremental, 1- absolute)
  - bit 1 - machine coordinates (0- program coordinates, 1- machine coordinates)
  - bit 7 - delayed start
- **axes_mask**
  - bit 0 - X axis
  - bit 1 - Y axis
  - bit 2 - Z axis
  - bit 3 - A axis
  - bit 4 - B axis
  - bit 5 - C axis
- **distance** - distance to go - integer value in 0.01 units (mm or inch depends on CNC setup)

NOTE: Use the bit conversion:

bit	converted value
bit 0	0x01
bit 1	0x02
bit 2	0x04
bit 3	0x08
bit 4	0x10
bit 5	0x20

Examples:

```
g0moveA(0,1,1000);// Move axis X to 10mm from current position (incremental)
g0moveA(1,4,0);// Move axis Z to Position =0 (absolute)
```

Easy to see that only 1 distance value programmed in **g0moveA** procedure. **Delayed start** is used to program motion in several coordinates. Delayed start bit tells to Motion controller to not start motion but just store distance position for future motion. Distance for several axes can be setup with delayed start bit, then the last procedure with no delayed start will start motion in all programmed positions.

Example

```
 //need to move to absolute position (100,50,20)
g0moveA(0x81,1,10000);// Set X=100 (absolute, delayed start)
g0moveA(0x81,2,5000);// Set Y=50 (absolute, delayed start)
g0moveA(0x81,4,2000);// Set Z=20 (absolute, delayed start)
g0moveA(1,7,0);      // Start move to (100,50,20)
```

Motion command is asynchronous operation. Actual motion is started a few milliseconds after g0moveA code executed. PLC procedure usually should wait motion is finished or monitor some event (sensor triggered) while motion executed.

NOTE: The drawback of this motion method is that after receiving the g0move command, the controller sends the movement values (distance, speed, acceleration) to the myCNC software, after which it then receives the recalculated values from the software in the units the controller requires (pulses, etc). As can be noted, this requires some additional time for the back and forth sending of information, which results in delays of up to 150ms (or even higher if using a Windows PC). If it is necessary to circumvent this delay, please use Option 2 for motion commands (described below).

A PLC procedure can obtain the current Motion Control State. Current Motion Control State is mapped to global variables array variable #6060. Return values while reading the variable are

Value	Description
0	Positioning motion (G0 code/ g0moveA running)
1	Line interpolation motion (G1 code)
2	Arc interpolation motion(G2/G3 code)
0x4d (symbol 'M')	Tech code runnunig (M-code/PLC procedure)
0x57 (symbol 'W')	Wait/Idle mode

Example (Move Z up to 10mm, wait till finished): Example

```
move_up=10;
g0moveA(0,4,move_up*100);  //Z axis;
timer=200; do{timer--;}while (timer>0);
 //wait 10ms , motion should start, motion state will turn to **0**
do{ code=gvarget(6060); } while(code==0);
```

Global variable #7080 is mapped to positioning speed register. Writing to this register will change motion speed for future **g0moveA** calls. Speed value is integer value given in units per minute.

Example (setup positioning speed for g0moveA calls)

```
gvarset(7080,3000);//will set positioning speed to 30000 [mm/min]
```

M02 procedure handler (which executed at the end of g-code file or when **Stop** button pressed) may contain lift spindle up before turning it OFF to prevent router bits overheat. A complete example of M02.plc is shown below

M02.plc

```
#include pins.h
#include vars.h
```

```
main()
{
  timer=0;

  if (proc==plc_proc_spindle)
  {
    if (lift_up>1)
    {
      gvarset(7080,speed_z);//set speed
      g0moveA(absolute,4,lift_up*100);//absolute programming; Z axis;
      timer=200; do{timer--;}while (timer>0);//wait till motion started
      do { code=gvarget(6060); } while(code==0);      //wait till
motion is stopped
    };
  };

  portclr(OUTPUT_MIST);
  portclr(OUTPUT_FLOOD);
  gvarset(7372,0);//Reset Mist State
  gvarset(7373,0);//Reset Flood State

  dac01=0x0;

  portclr(OUTPUT_SPINDLE);
  portclr(OUTPUT_CCW_SPINDLE);
  gvarset(7370,0);//Spindle State
  gvarset(7371,0);//Spindle Speed Mirror register

  command=PLC_MESSAGE_SPINDLE_SPEED_CHANGED;
  parameter=0;
  message=PLCCMD_REPLY_TO_MYCNC;
  timer=10; do{timer--;}while (timer>0); //pause to push the message
with Spindle Speed data

  proc=plc_proc_idle;
  message=PLCCMD_MOTION_ABORT;
  exit(99);
};
```

NOTE: Note the `proc=plc_proc_idle` code at the end of the M02 macro. If this line is not present, then the Stop command is not completed, and the system will remain suspended as it waits for the PLC code to complete all operations. As a result, running/restarting the control program will not work correctly.

First block of this procedure check if spindle currently is O, and **lift_up** variable is positive value and the lift tool up for given value set in plc-variables.xml configuration file. For those who don't need this lifting can remove the lines and rebuild PLC (see rebuild buttons in PLC Builder interface)

Option 2

NOTE: At the time of writing this manual, Option 2 for motion control is available in the Testing branch of myCNC firmware (version 15,050 and above). For firmware update instructions, please consult the manual for your particular controller.

If bit #13 is set for the axis mask, then Option 2 of the PLC motion command is used (instead of Option 1, described above). In this case the control board itself will handle all calculations necessary for the positioning motion without the support of myCNC software. As a result, the controller does not need to communicate with the Host PC and the movement will be started immediately (unlike Option 1, where extra communication with the Host PC leads to an additional delay of about 100-150ms).

The drawback of this method is that the movement instructions are programmed in **pulses** rather than the more conventional units of mm/inches, as the controller is not aware of the motion units that are set in the myCNC software.

Useful global variable registers:

- #8630 - used to set the motion speed, in pulses/second
- #8631 - used to set the ramp-up time to a given speed, in ms

Example of a motion command for this method:

```
g0moveA(0x01,0x1001,16000);           //absolute programming; X axis;
```

- 0x01 - Absolute coordinates
- 0x1001 - bit #13 (0x1000) and X-axis mask (0x01)
- 16000 - coordinate for the selected X-axis, in pulses (for example, if the pulse-per-mm value for the X-axis is equal to 800, the movement will be equal 1600/800=20mm

draw_square

```
wait_motion_end()
{
  timer=2; do{timer--;}while (timer>0);  //wait motion started
  do
   {
      ex=0; code=gvarget(6060);
      if (code==0x4d) {ex=1;};
      if (code==0x57) {ex=1;};
    } while(ex==0);
};

square()
{
  gvarset(8630,50000); //speed/frequency 50kHz
  gvarset(8631,100);   //Time ~ 0.1sec (in milliseconds)

  g0moveA(0x01,0x1001,16000);        //absolute programming; Y axis;
  wait_motion_end();
  g0moveA(0x01,0x1002,16000);        //absolute programming; Y axis;
```

```
  wait_motion_end();
  g0moveA(0x01,0x1001,8000);          //absolute programming; X axis;
  wait_motion_end();
  g0moveA(0x01,0x1002,8000);          //absolute programming; Y axis;
  wait_motion_end();


};
```