EuroForth 2004

Experiments in real time control in Windows using Forth

N.J. Nelson B.Sc C.Eng. M.I.E.E.
Micross Automation Systems Ltd.
Units 4-5, Great Western Court,
Ross-on-Wye,
Herefordshire
HR9 7XP
Tel. +44 1989 768080
Fax. +44 1989 768163
Email njn@micross.co.uk

Abstract.
The award of a contract stipulating a specific type of hardware necessitated the development of some new technology in the form of a Forth Soft PLC to interface with the specified hardware in order to operate real time control.

1. Introduction

For some years we have been producing automation systems for commercial laundries based on the following software structure:
HMI in Windows
Device driver
RTX card processor
IX1 fieldbus I/O processor, to component I/O using Interbus S

All of these elements except the device driver were implemented in Forth.

In July of this year we were offered a valuable and interesting contract which however stipulated the use of PLCs by the Austrian company Bernecker & Rainer (B & R). These devices are widely used in the laundry industry but do not offer Interbus S, and therefore our usual RTX Virtual PLC card could not be used, and some new technology would have to be developed.

The contract specified a startup date of 29th September, and required a good deal of what Stephen Pelc calls “Russian Front Programming”.

2. System overview

Guided by the B&R UK technical sales department, we started work initially on the following structure:
HMI in Windows XP (minimum possible modifications to our existing program)
B & R soft PLC
B & R Powerlink interface card
Powerlink I/O modules

Powerlink is an Ethernet based protocol which imposes real time deterministic communications on top of a 100Mb standard hardware structure. Unfortunately the protocol is so complex that a powerful coprocessor (usually an ARM) is required to handle it.

After 6 weeks of struggle, we were obliged to give up on the soft PLC when B & R finally admitted that it didn't do what it said on the box. Instead we substituted a rack-based PLC for the soft PLC, and implemented a dedicated UDP connection between that and the PC.

3. Brief overview of the traditional differences between a PLC and a PC

PLC	**PC**
1. Deterministic	Statistical
2. Simple software	Complex software
3. Non volatile	Volatile
4. Generally primitive development environment	Sophisticated development tools
5. Highly predictable and reliable	Uncertain.

4. The B & R PLC

The B & R PLC differed from a standard PLC in a number of respects.
1. It was deterministic until one tried to overload a time slice, whereupon it fell over unpredictably
2. It had quite complex software
3. It was only partially non volatile
4. It had quite a sophisticated development environment
5. It had a wide range of interesting bugs and quirks which pushed it distinctly in the "uncertain" direction

An important reason for its oddities was no doubt due to it being based on an Intel PC type of hardware.

In addition to the standard IEC 6-1131 programming “languages” (see my previous paper "Industrial Control languages: Forth vs. IEC61131", EuroForth 2000), it offered both C and Basic – most unusual in a PLC.

One approach that we considered was to port our standard RTX Forth control software into B & R Automation Basic. We decided against this for two reasons. Firstly, one of our competitors had attempted a similar method before and failed in spectacular and costly fashion. Secondly, we had already wasted 6 weeks on a very tight schedule, and there was simply not enough time left.

Instead, we adopted a completely new approach. We divided the PLC responsibilities into two sections according to how time-critical they were. The strictly time-critical functions were implemented in Automation Basic on the B & R PLC. Those functions which required a prompt but less critical response were implemented in a soft PLC written in Forth and running in a very high priority task under Windows.

5. Analysis of time-critical functions

It frequently turns out that only a small number of the control tasks for a piece of equipment are genuinely time critical, and this proved to be true here. Most of the equipment to be controlled consisted of a complex layout of gravity driven suspended monorail. It is essential to make sure that you don't miss any position detecting proximity switches as the containers are running along the tracks. It's easy to calculate from the maximum speed of of the container and the dimensions of the proximity switch and sensing plate that a 20ms scan time is required. However, it is rarely necessary to actually respond to the position detector in real time. Accordingly we adopted our usual approach of implementing counters on every logical input. Since this can be done in a loop, only a few lines of Basic code are needed. There are a few other genuinely real time requirements. One is where the containers pass from a single track one wheel behind the other section, to a double track one wheel on each track section. Here you have to operate a set of points while the container is straddling them. Again this takes only a couple of lines of Basic code. The end result of this analysis was an absolute minimum of code to be implemented in Basic.

6. The "Time Important" functions

This simply left us with the task of implementing a time-important (but not “critical”) soft PLC within Windows. The sort of thing I mean by time-important is for example if one is flashing a lamp with a period of 1s, an operator should not notice any erratic operation. I decided that a scan time of 50ms would be needed to achieve this, and started to investigate whether this could be achieved reliably in Windows.

7. Different types of soft PLC

7.1 Using a Real Time Operating System

Here the RTOS installs first and runs beneath Windows. It allows Windows to run in predetermined time slices. It is possible to make a true deterministic control system on a PC using this technique.
This was the approach used by the B & R soft PLC, and the reason for its downfall. The major problem with this approach is that it needs to interface with hardware which is not on the motherboard:

a) Non volatile memory
b) An I/O device – in this case a Powerlink ARM controller

In the case of B & R, these were both implemented on a single PCI card.
But of course since one is running *beneath* Windows, one can't use the Windows driver structure to communicate with the card and as a result hardware independence is lost. The B & R soft PLC runs on their own range of IPCs. It does not work on an absolutely standard HP-Compaq desktop PC. But B & R only believed this when we shipped our entire development environment to them in Austria.

We could not use B & R's own IPCs for the following reasons:

a) Highest performance is 133MHz
b) Five times the price of a 2.8GHz desktop
c) Suspect availability of spares in the future (5 years time)
d) Only runs XP embedded, not XP Pro

7.2 Using a Windows driver program

Well, it must work, mustn't it? After all, you can run a serial communications link at a ridiculously high baud rate with only a 16 byte FIFO as hardware support, and it doesn't miss a trick. I have been intending for years to write a version of Forth which runs in a driver environment, but I didn't have time (again).

7.3 Using a high priority task within Windows itself

It is not possible to pretend that Windows has a real-time multitasking system. However, with 2.8GHz of horsepower behind you, surely something fairly quick on the draw can be arranged.

8. Designing the Forth Soft PLC

a) Setting the task priorities

Task priority is used by Windows to decide which task should be alloted time to execute. There are 32 priority levels. A task will only execute if no task with a higher priority level is due to run.
In Windows, a two level approach is used to set the priority level.
There is the base priority of the process, and the relative priority of the tasks within that process.
Is it easy to experiment with the effect of each using for example:

```
WINGETCURRENTPROCESS HIGH_PRIORITY_CLASS WINSETPRIORITYCLASS DROP
```

and

```
WINGETCURRENTTHREAD THREAD_PRIORITY_ABOVE_NORMAL WINSETTHREADPRIORITY DROP
```

If both of these are set to "_NORMAL_" then a thread runs a level 7 if it's in the background and level 9 if it's in the foreground.

There are lots of tempting possibilities, including REALTIME_PRIORITY_CLASS. However, the minimum thread priority that can be associated with this class results in an overall priority of 16. This is higher than any normal Windows application. Now, our own application visualises a highly dynamic system, and paint messages are constantly being sent to the main window. As a result, any application whose main thread is at a lower priority is effectively locked out. The combination we therefore arrived at was NORMAL for the class, and TIME_CRITICAL for the soft PLC thread, which results in an overall priority of 15. This is higher than any application, but lower than any of the system tasks (disk flushing etc.).

b) Setting the scan time

We determined earlier that a 50ms scan time was required to obtain the desired responsiveness. However, it is not possible to instruct a thread to execute at exactly specified intervals. Instead, the Windows SLEEP function (wrapped to Forth as WAIT) instructs the system not to call the thread again for a minimum of the specified time in milliseconds. The exact value cannot be determined in advance – it depends on how long the thread takes to execute, and how long it takes before the system reschedules the ready thread for another time slice. We therefore used a little feedback system to measure the actual scantime and adjust the WAIT time as required.

Here is the soft PLC thread

```
TASK: BANDR       \ B & R communications and soft PLC thread
  WINGETCURRENTTHREAD                    \ Set priority of thread
  THREAD_PRIORITY_TIME_CRITICAL
  WINSETTHREADPRIORITY DROP
  INVALID_SOCKET BANDRSOCK !             \ Set socket to invalid
  TRACKNET-WINDOW HANDLE UM_SETDIAG      \ Set starting up diagnostic
  19 0 WPMD
  PC-INIT                                \ Initialise coms with main thread
  ASCINIT                                \ Initialise analog to serial comms
  TASKASSESS-INIT                        \ Initialise task performance
  50 WAITTIM !                           \ Start with nominal 50ms wait time
  BEGIN
    TASKASSESS                           \ Performance indicators
    BANDRMAIN                            \ The SPLC main loop
    LOOP-ADJUST                          \ Adjust wait time to obtain 50ms
    WAITTIM @ WAIT                       \ Allow other threads
  AGAIN
;
```

and here is the feedback function

```
: LOOP-ADJUST ( --- ) \ Adjust wait time to obtain 50ms average scan
  1 LOOPCTR +!                           \ Increment loop counter
  LOOPCTR @ 9 U> IF                      \ Done 10 loops
    0 LOOPCTR !                          \ Reset counter
    WINGETTICKCOUNT                      \ Get time in ms
    DUP OLDMS @ - ABS                    \ Get time difference
    500 U> IF -1 ELSE 1 THEN WAITTIM @ +  \ Adjust wait time
    20 MAX 50 MIN WAITTIM !              \ Limit
    OLDMS !                              \ Save for next time
  THEN
;
```

The measured scantime is averaged over 10 cycles before applying feedback. In the final application, WAITTIM homed in to 47ms, and never deviated by more than 1ms. In other words, the criterion that no visible erratic operation of a flashing lamp with a period of 1s can be noticed, is easily met.

The task assessment tool in its simplest form just keeps a note of the longest and shortest actual scantimes (we also used a more sophisticated tool which recorded individual scans over a period).

```
: TASKASSESS { | curtick difftick -- } \ Maintain task performance indicators
  WINGETTICKCOUNT -> curtick
  LASTTICK @ IF
    curtick LASTTICK @ - -> difftick
    difftick MINTIME @ MIN MINTIME !
    difftick MAXTIME @ MAX MAXTIME !
  THEN
  curtick LASTTICK !
;
```

9. The results - how realtime?

The basic task ran with surprising accuracy. Over intense tests on the PC, MINTIME was never less that 46ms and MAXTIME never exceeded 63ms. These figures were with a task containing dummy time-consuming loops.

In the real case, however, the UDP Ethernet communications must be included in the task. Although the PLC UDP link is provided with its own network card, the introduction of the UDP processing into the task has a highly detrimental effect on the jitter. If other network communication is taking place on the other card (and particularly if invalid connections are being searched for) the MAXTIME can rise to 150ms. The conclusion must be that Windows uses a single task to control all sockets on a system, and therefore a delay on one locks out all others. There is not much we can do about this. The jitter affects only very occasional cycles and therefore our flashing lamp still has no human-discernible erratic behavior.

10. Conclusion - Did it work?

It worked with remarkable success and is installed in a very large laundry less than 100 miles from the conference venue. Some system statistics:

- 5 PCs
- 33 other HMIs
- 3 Master PLCs
- Approx. 70,000 lines of Forth, less than 200 lines of Basic
- More than 2000 logical inputs and outputs
- 36 analog inputs
- 28 CANbus connections to other equipment
- 1500 loads of washing simultaneously visualised
- Over 200 mechanical tasks controlled simultaneously in real time
- Production handled currently approx. 50 tons per 10 hour day and rising

NJN
November 2004