

Tools to work on UCB

Revision	Date	history
rev. a	22/12/10	Starting revision

General description

UCB is a microcontroller based machine controller, build around an LPC2368 device, ARM-based NXP processor.

UCB includes:

- local power supply, starting from 24V down to 5V and 3,3V
- 1A power output, protected against short circuits, to run DC motor matrix up to 16x8 or 14x10
- 8 general inputs (to connect to input devices)
- 8 generic outputs
- keyboards interface (up to 16x8 matrix or analog keypad)
- display interface
- modem serial port
- Zigbee socket
- aux serial port
- MDB serial port
- provision for Ethernet connectivity
- SD/MMC interface
- USB device

LPC2368 will include 58kB of RAM and 512kB of segmented flash memory. The segmented flash can be programmed by run-time.

Firmware

Firmware is build in C language, using tools for ARM made by IAR.

Actual software is compiled using IAR Embedded Workbench rev 4.41A.

No operating system is used, only a basic scheduler to handle some basic tasks.

Time critical tasks are handled directly as interrupt tasks.

A boot loader is provided : the boot loader is always available, loaded on some protected flash segments, and running as a default tasks after machine start-up.

Boot Loader

Boot loader is a piece of firmware always available on the machine. After the processor is reset, the boot loader is running, checking for any available firmware update file on SD/MMC card.

If any available, this is loaded into the processor main flash.

If no update available, the main application loaded into flash is checked, and if ok is made to run.

There is no provision for dynamic load of libraries, the full code must be statically linked before loading.

Main application

Based on some basic independent tasks

- selection operation
- motor drivers (including motor availability check and homing)
- display driver
- keyboard driver
- DEX handler
- MDB basic protocol handler
- MDB peripherals specific device drivers
- Modem device driver
- SD/MMC device driver, handling a FAT file system
- drop sensor driver
- graphic display driver

This task will share the single memory address space of the processor, because no memory protection is available.

They share the data memory, but the main communication structure between the different tasks is event based, so new tasks can be added to use the same communication mechanism with no modification of the existing task code.

Main application has some parts which depend on the machine type where the UCB is installed: this conditional code execution will depend on a machine ID that is stored in a memory soldered to the harness board (which is unique for each machine type).

There is no structured data base on board. Data is stored in compiler defined structures.

Following and example of the API available on UART ports

```
byte commCfgPort( byte _ch, LPC_Uart_Baud_t _baud, LPC_Uart_WordLenth_t _bits, LPC_Uart_ParitySelect_t
 _parity, LPC_Uart_StopBit_t _stops );
byte commRxFlash( byte _ch );
byte commRxIntDis( byte _ch );
byte commRxIntEn( byte _ch );
byte commTxIntDis( byte _ch );
byte commTxIntEn( byte _ch );
```

```
byte commGetChar( byte _ch, byte *_data );  
byte commPutChar( byte _ch, byte _data );  
void commPutStr( byte _ch, byte *_str );  
BOOL commIsEmpty( byte _ch );  
BOOL commIsFull( byte _ch );  
BOOL UART_readTxBuf( _RING_BUF_*_p, byte *_txReg );
```

Adding additional indipendent applications

To have the capability to add additional indipendent applications we need to define:

- a static address range available to the new application, in terms of RAM and Flash
- a static table of API from our main application and the new application
- the list of available events both application can share

We can modify the boot loader to handle two different (and indipendent) applications, each application interfacing with the other via the static table of Api references and the events.