Happyboard

User Guide
0.61

ross@glashan.net

January 6, 2008

Contents

About

The Happyboard is a feature-packed robot controller board specifically designed for 6.270. This manual provides a brief overview of the boards functionality, and covers how to set up and develop code for the Happyboard.

Features

ATMega128L Processor

The Happyboard is powered by an Atmel AVR ATMega128L 8bit RISC processor, running at 8MHz. The ATMega has 128KB onboard flash and 4KB onboard RAM memory.

32KB External RAM, 128KB External flash

The processor is connected to an external 32KB of SRAM which is available to user code. There is also an external 128KB flash chip for non-volatile data storage (it is also used for configuration parameters and FPGA code).

Spartan 3 XC3S50 FPGA

The processor is also connected to a memory-mapped Spartan 3 FPGA, which controls most of the IO on the Happyboard. The FPGA offloads the driving of the motors, servos, shaft encoders and digital inputs from the CPU, leaving more processing time for user code.

6 2A Motor ports

The Happyboard has 6 Motor ports each capable of driving a motor at up to 2A. Each motor port is also current-sensed and has two LEDs to indicate motor power and direction

6 RC Servo ports

Alongside the motors are 6 servo ports. These ports allow the position control of standard RC servos.

8 digital inputs

8 buffered, digital inputs allow the reading of digital sensors like bump switches.

16 10bit analog inputs

16 analog inputs are read by MCP3008 ADCs which provide 10bit readings of analog sensors like photodiodes, distance sensors and gyros.

4 shaft encoder inputs

4 counter inputs allow for the measurement of sensors like shaft-encoders to calculate shaft rotation distance, velocity, etc.

16x2 LCD, Pushbuttons, Buzzer, and Frobknob

The Happyboard has a 16x2 character LCD display for printing information to the user. It also features 2 push buttons and a ”Frobknob” wheel allowing the user to make simple user interfaces. A small piezo beeper allows for basic sound and music playback.

USB

A usb mini-b connector provides a simple, fast interface for programming the Happyboard. The Happyboard is automatically recognised as a USB serial port by modern operating systems.

JTAG Debug Interface

A small 8 pin connector provides JTAG access to the processor. This interface, combined with some simple hardware allow for powerful low-level debugging access to the processor.

I^2C Expansion connector

I^2C is a commonly used 2-wire multi-device bus, and the Happyboard provides a expansion header for adding external I^2C devices.

Power Supply

The Happyboard can be powered by any 7.5V - 9.5V battery or power supply. The on-board regulators generate a variety of voltages used by the different portions of the Happyboard: 5V@3A for servos, 5V@0.5A for user inputs, and 3.3V@1A for logic.

Board Connectors

The diagram below shows all the connectors and ports on the Happyboard.

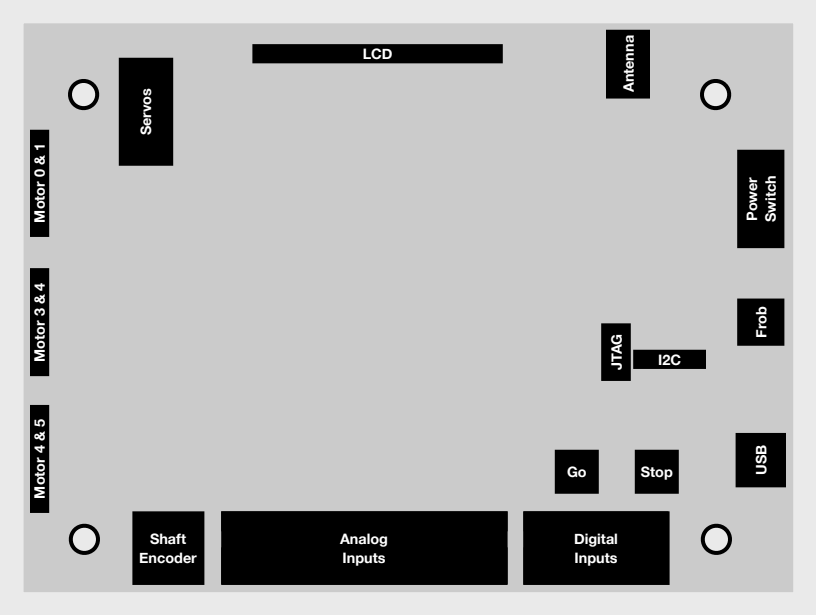
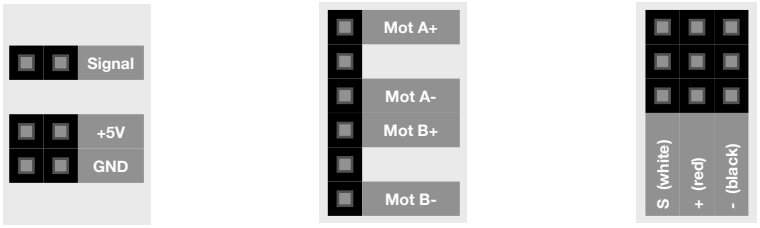


Figure 1: *Happyboard Ports & Connectors*



(a) Inputs (b) Motors (c) Servos

Figure 2: *Port Pinouts*

Block Diagram

The diagram below shows the major components of the Happyboard and their interconnections.

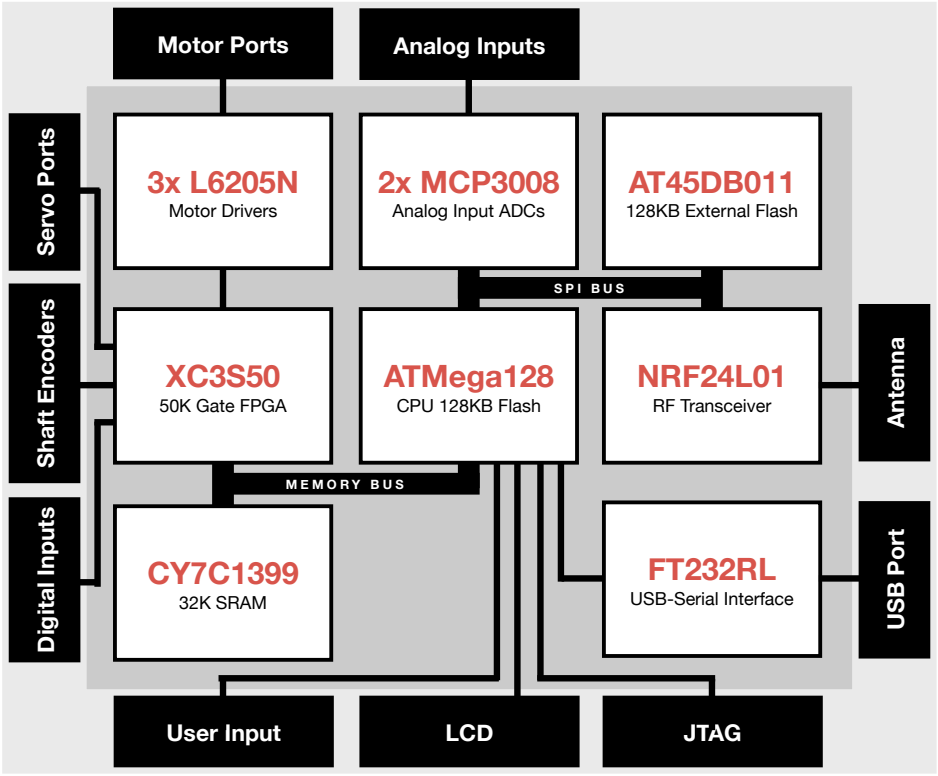


Figure 3: *Happyboard Block Diagram*

Quickstart

This section describes how to quickly get up and running developming code for the Happyboard.

Requirements

Before starting, the following will be required:

- Happyboard
- 7.5-9.5V battery or power supply
- USB Mini-B cable
- Windows, Mac or Linux computer
- Small jumper wire or sensor for testing
- DC motor and RC Servo for testing

Happyboard Setup

We should check that the Happyboard is operating correctly before connecting it to the computer. Refer to Figure 1 for the location and pinout of the various connectors and ports.

1. **Connect the battery or power supply to the Happyboard.** Ensure the polarity of the input power is correct, as the Happyboard does not have reverse-polarity protection. The positive terminal of the power connector is to the right of the board.
2. **Power on the board.** The ON position of the switch is away from the frob knob.
3. **Check that the board boots correctly.** The Happyboard should boot up into Happytest (unless it has already been reprogrammed) and print ”Happytest v0.6”.

4. **Start Happytest.** Press GO to start the first test.

5. **Test the servo drivers.** Plug a servo into one of the servo ports (see Figure 2(c) for servo pinout). Use the frob knob to change the servo angle.

6. **Test the motor drivers.** Plug a DC motor into a motor port (see Figure 2(b) for motor pinout). Use the frob knob to change the motor velocity and direction. Press GO to switch between motor ports.

7. **Test the digital inputs.** Plug a jumper wire into a digital port (see Figure 2(a) for pinout). A digital input pulled to ground will show a 1 on the screen, 0 otherwise.

8. **Test the analog inputs.** Plug a jumper wire or sensor into an analog port (see Figure 2(a) for pinout). Use the frob to choose which analog port to read. The Reading will vary between 0 and 1023 representing a voltage between 0 and 5 volts on the input.

9. **Test the encoder inputs.** Plug a jumper wire or sensor into an encoder port (see Figure 2(a) for pinout). For each transition of the encoder input the encoder value will increment.

Computer Setup

Happyboard development is possible on most modern operating systems, so this document will cover setup and development with Linux, Mac OS X, and Windows.

Before installing any development tools drivers for communicating with the Happyboard are required. These drivers are available for most operating systems as the Happyboard uses a common USB chip from FTDI. For drivers for Windows and Mac OS X check the FTDI Site. For most Linux distributions, the drivers should be installed by default.

With drivers installed, check that the Happyboard is recognised by plugging it into a USB port on the computer and powering the board up. The Happyboard should be recognised as a ”Happyboard” or a ”USB Serial Port”.

The basic development system for the Happyboard is based around a GNU toolchain, including the usual C tools (GCC, the GNU linker, assmebler, etc), as well as a AVR specific C library. A version of these tools is available for most major platforms. Below is a quick outline of setting up the tools.

Linux

Setting up the AVR tools on Linux is highly dependent on the specific linux distribution used. For Debian or Debian-based distributions (Ubuntu, etc) the following is required:

```
sudo apt-get install binutils-avr gcc-avr avr-libc avrdude
```

For Fedora or other RPM-based distributions, the following should work:

```
sudo yum install avr-binutils avr-gcc avr-libc avrdude
```

Once installed check avr-gcc (at least GCC version 3.3 is required) and avrdude work correctly:

```
avr-gcc --version
avrdude
```

Mac OS X

A simple pre-packaged version of the AVR tools is not available for Mac OS X, but they are available through the Fink or MacPorts distributions. If you have either installed, a few simple terminal commands allow for the installation of the tools. For MacPorts:

```
sudo port install avr-gcc avr-binutils avr-libc avrdude
```

For Fink:

```
sudo apt-get install avr-gcc avr-binutils avr-libc avrdude
```

Once installed check avr-gcc (at least GCC version 3.3 is required) and avrdude work correctly:

```
avr-gcc --version
avrdude
```

Windows

For windows, a precompiled package of all the tools is available as WinAVR. Simply download the latest WinAVR package and install with all the defaults.

Compiling

With the development tools installed, you need the 6.270 libraries to start writing robot code. Download the latest version of the code from the 6.270 Contestants Page.

Extract the 6.270 libraries to a convenient location which will create a 6.270 directory. The 6.270 directory contains the following subdirectories:

- **doc**/ contains documentation relating to 6.270 code.
- **inc**/ contains header files for JoyOS and Happylib.

- **lib**/ contains JoyOS and Happylib binary library files.
- **src**/ contains example robot projects.

Open a terminal (Cmd.exe on windows, Terminal.app on Mac OS X) and change to the src/robt directory, and run Make.

```
cd <path-to-6.270>/src/robot
make
```

The example project (consisting of a single C file, umain.c) should be compiled and the following displayed:

```
Compiling umain.o
Linking robot.elf
Generating hex file robot.hex
```

A robot.hex file should now exist, ready to be downloaded to the Happyboard.

Downloading and Running

Before code can be downloaded to the Happyboard, we need to configure the port used to perform the download. The serial port name varies between operating systems and computers, so instructions on finding the correct port are shown below. Once you have the correct port number, open the Makefile in the robot directory and change the PORT line to the correct port name.

Linux

On most linux distributions the Happyboard will show up as a serial port at `/dev/ttyS**`. To find the correct port number, plug the happyboard into a USB port and turn it on. After a few seconds run the dmesg in a terminal. This should display the detection on driver setup messages for the Happyboard serial port. Example dmesg output shown below:

```
FIXME
```

Mac OS X

On Mac OS X the Happyboard will show up as a serial port at `/dev/tty.usb**`. To find the correct port number, plug the happyboard into a USB port and turn it on. After a few seconds run the following command in a terminal. This should display the correct port name.

```
ls /dev/tty.usb*
```

Windows

On Windows the Happyboard will show up as a serial port named `COM*`. The correct COM port number can be determined in the Device Manager. Right click on ”My Computer”, click ”Properties”, then ”Hardware”, then ”Device Manager”. Under the ”Ports (COM & LPT)” section, the last COM port should be the Happyboard.

Performing the Download

Once the Happyboard port is correctly configured in the Makefile, we’re ready to download code to the Happyboard. Put the Happyboard into download mode by holding the STOP button down while turning the Happyboard on. After a few seconds release the STOP button. The Happyboard should now display ”Happyboot”, indicating it is ready to accept code downloads. In the same directory as the robot.hex file run the following command:

```
make program
```

The code will be programmed to the Happyboard’s flash memory. Once the download completes, reboot the Happyboard (turn off, then on - do not hold down stop this time). The Happyboard should boot up and run the demo program which scrolls the word ”ROBOT” back and forth across the LCD.

JoyOS

The Happyboard runs a simple operating system called JoyOS. JoyOS provides a number of features designed to make building a robot with the Happyboard easier.

Features

Simple Hardware Interface

JoyOS Provides a simple interface to all of the hardware on the Happyboard. All of the IO functionality is availabe to the user in a simple and straight-foward interface.

Mutlithreading support

JoyOS provides multithreading support, allowing users to develop code for handling many tasks at once (especially handy for processing IO tasks in the background). JoyOS also provides other functionality useful in multithreaded code, including locks, thread forking, and ISR support functions.

Debugging support

JoyOS also provides a selection of functions to assist in debugging code on the Happyboard. A variety of logging and error handling support is included. Various checks are also performedd to ensure the happyboard is operating correctly.

JoyOS Structure

The diagram below shows the major components of JoyOS.

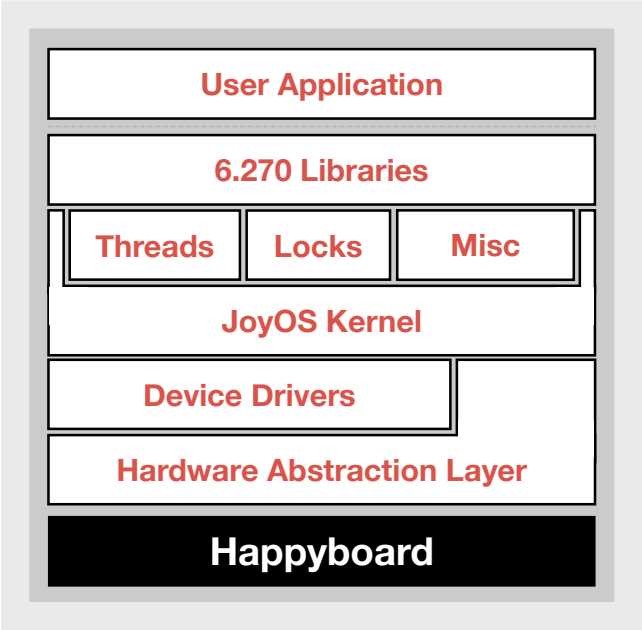


Figure 4: *Happyboard Block Diagram*

User Code Structure

User code to be run on JoyOS must have a few special functions to operate correctly.

- `void usetup()` - This function is called at the start of the 60 second calibration period at the beginning of a competition round.
- `void umain()` - This function is the main entry point for user code. JoyOS will create a new thread to begin executing this function.

For examples of JoyOS user code, see the src directory in the JoyOS code.

Function Quick-reference

Below is a listing of most useful function calls in JoyOS. For a more detailed explanation, see the JoyOS API reference.

Motor Ports

- `void motor_set_vel ( uint8_t motor, int16_t vel )`
  Set *motor* to velocity *vel*.
- `void motor_brake ( uint8_t motor )`
  Set *motor* to brake mode.
- `uint16_t motor_get_current_MA ( uint8_t motor )`
  Return the current drawn by *motor*.

Servo Ports

- `void servo_set_pos ( uint8_t servo, uint16_t pos )`
  Set *servo* to position *pos*.

Digital Ports

- `uint8_t digital_read ( uint8_t port )`
  Read the digital sensor on input *port*.

Analog Ports

- `uint16_t analog_read ( uint8_t port )`
  Read the analog sensor on input *port*.

Shaft Encoder Ports

- `uint16_t encoder_read ( uint8_t encoder )`
  Read the encoder count from *encoder*.
- `void encoder_reset ( uint8_t encoder )`
  Reset the encoder count for *encoder*.

Frob Knob & Pushbuttons

- `uint16_t frob_read ( )`
  Read the position of the frob knob.
- `void go_click ( )`
  Wait until the go button is clicked.
- `uint8_t go_press ( )`
  Check if the go button is pressed.

- `void stop_click ( )`
  Wait until the stop button is clicked.
- `uint8_t stop_press ( )`
  Check if the stop button is pressed.

Beeper

- `void beep ( uint16_t` *`freq`*`, uint16_t` *`duration`* `)`
  Beep *duration* milliseconds, at *freq* Hz.

LCD

- `int printf ( const char *`*`fmt`*`, ... )`
  Print *fmt* formatted text to the LCD.
- `void lcd_clear ( )`
  Clear the LCD.
- `void lcd_set_pos ( uint8_t` *`p`* `)`
  Move cursor to character *p* on the LCD.

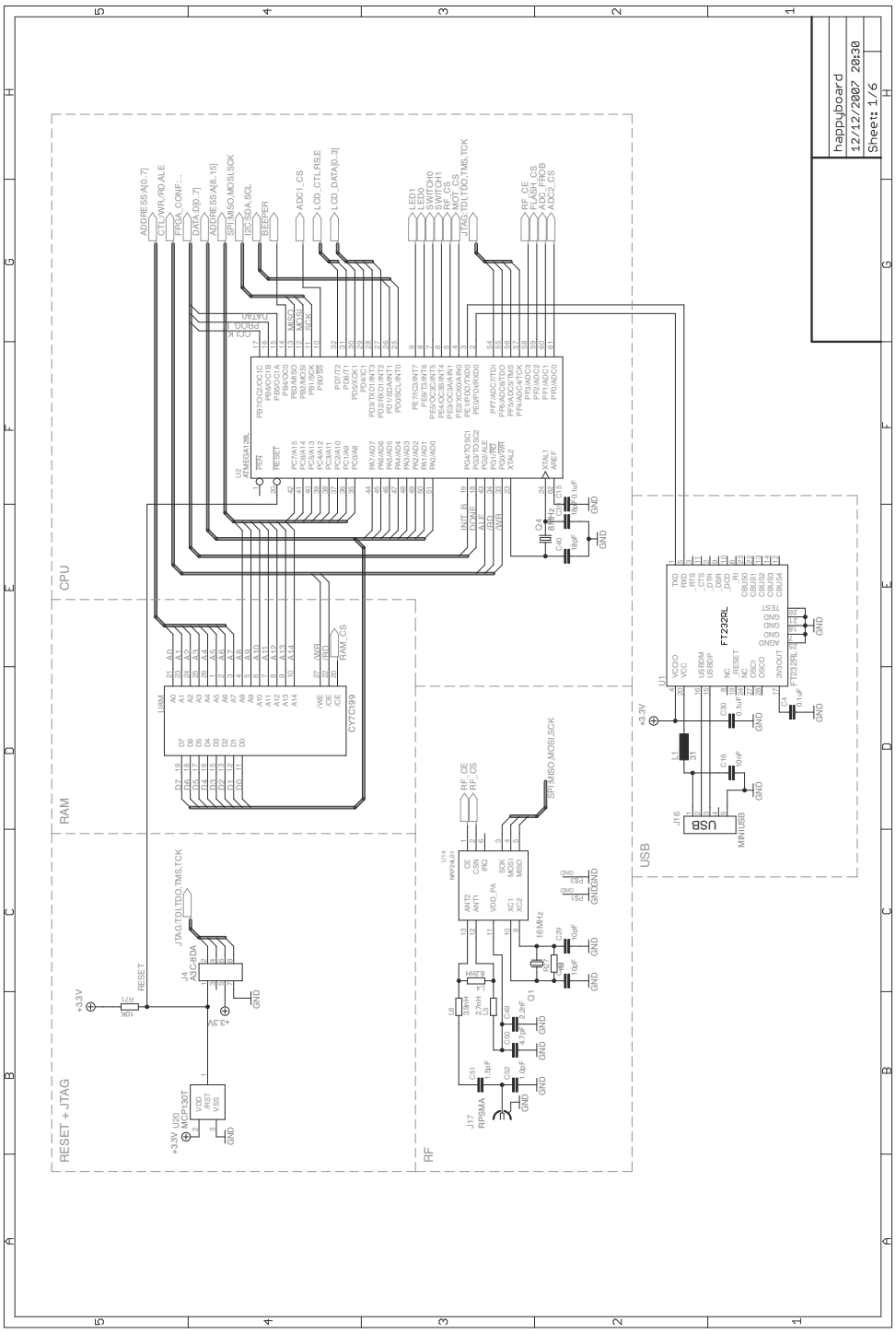
Threads

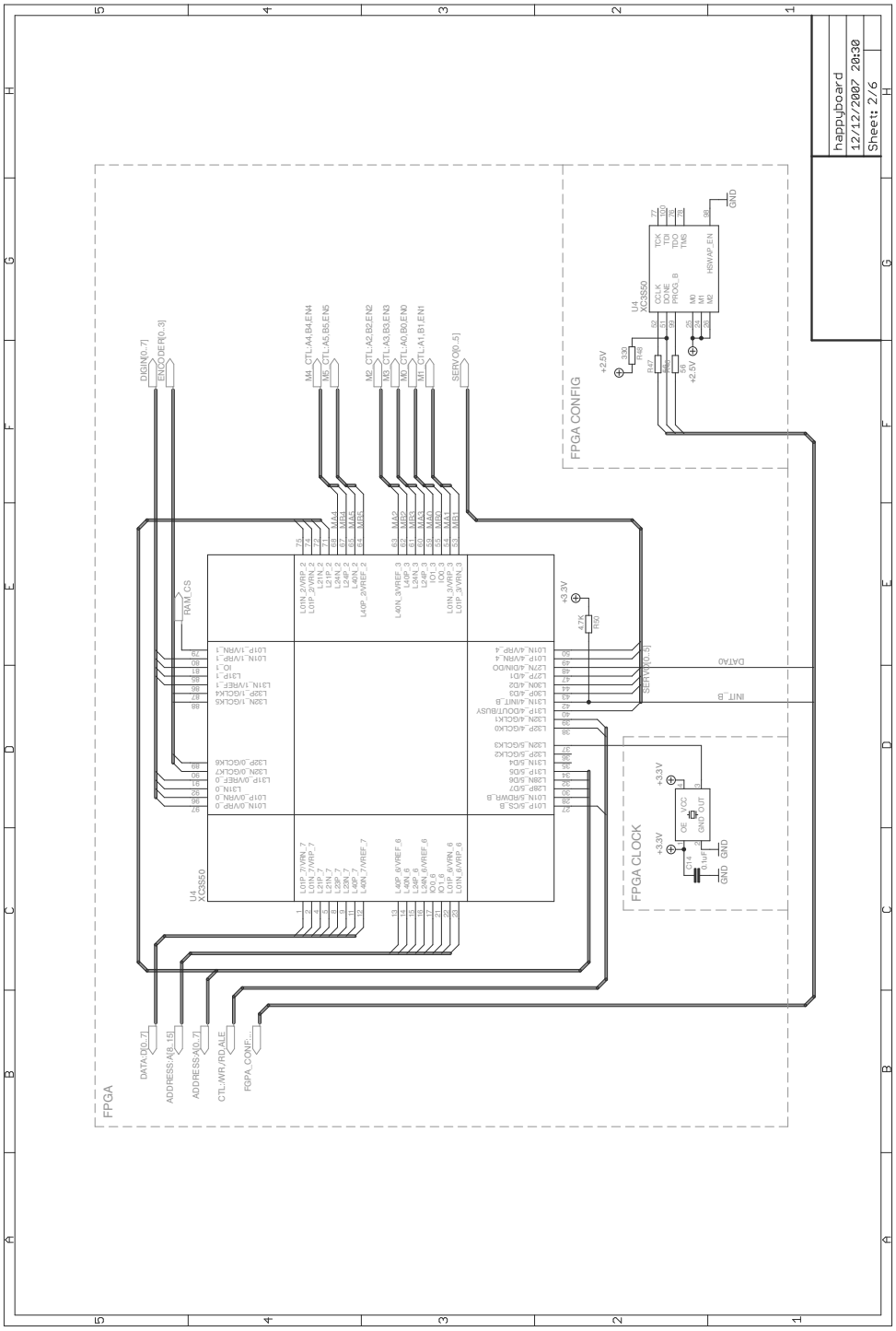
- `uint8_t create_thread ( int(*`*`func`*`)(), uint16_t` *`stacksize`*`, uint8_t` *`priority`*`, char *`*`name`* `)`
  Create a new thread that will execute *func* and then return. Allocate a stack of *stacksize* bytes, and set the thread *priority*. Also set a thread *name* for debugging.
- `void yield ( )`
  Give up the processor to another thread until rescheduled.
- `void pause ( uint16_t` *`ms`* `)`
  Pause a thread for *ms* milliseconds.
- `uint32_t get_time ( )`
  Return the number of milliseconds elapsed since startup.

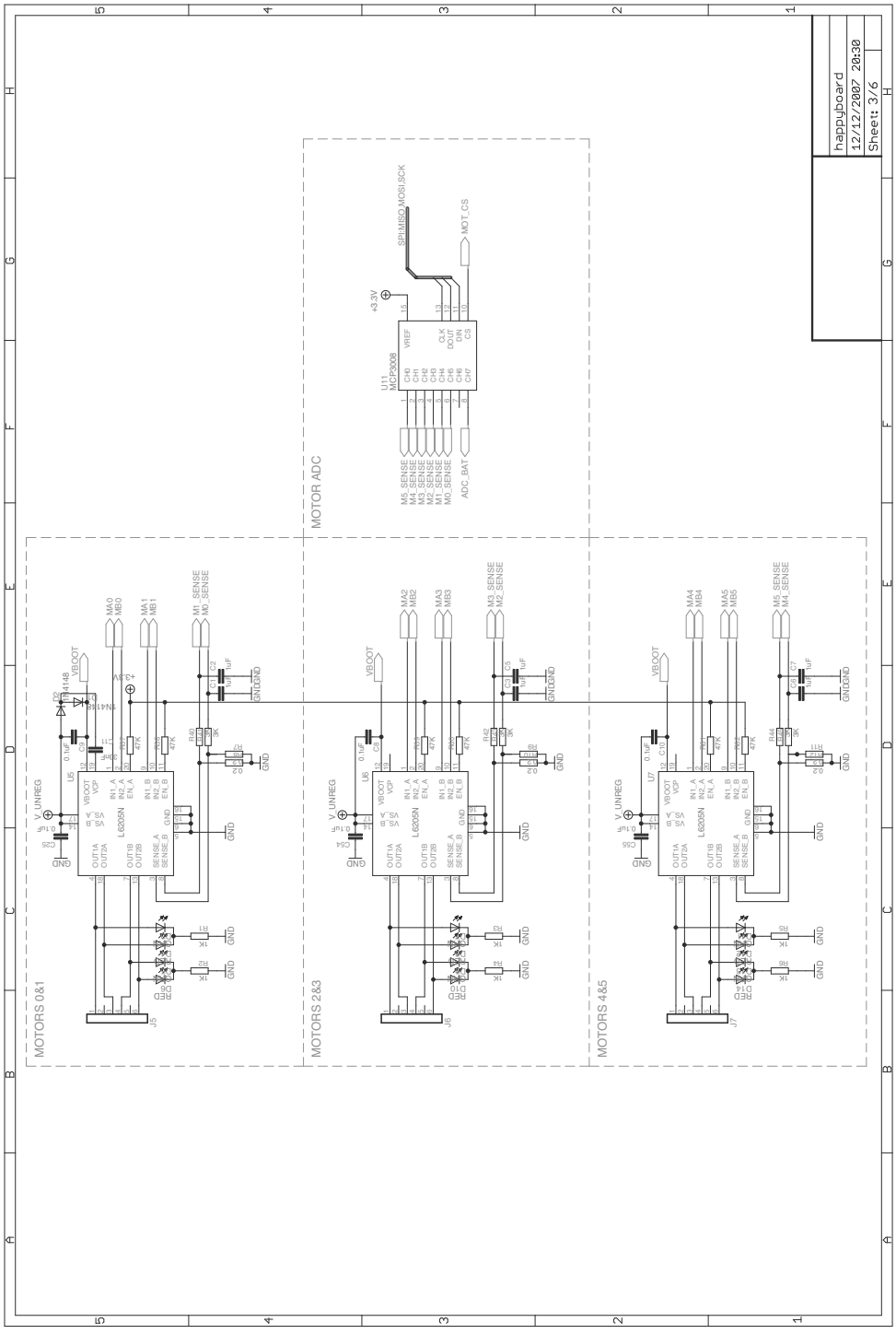
Locks

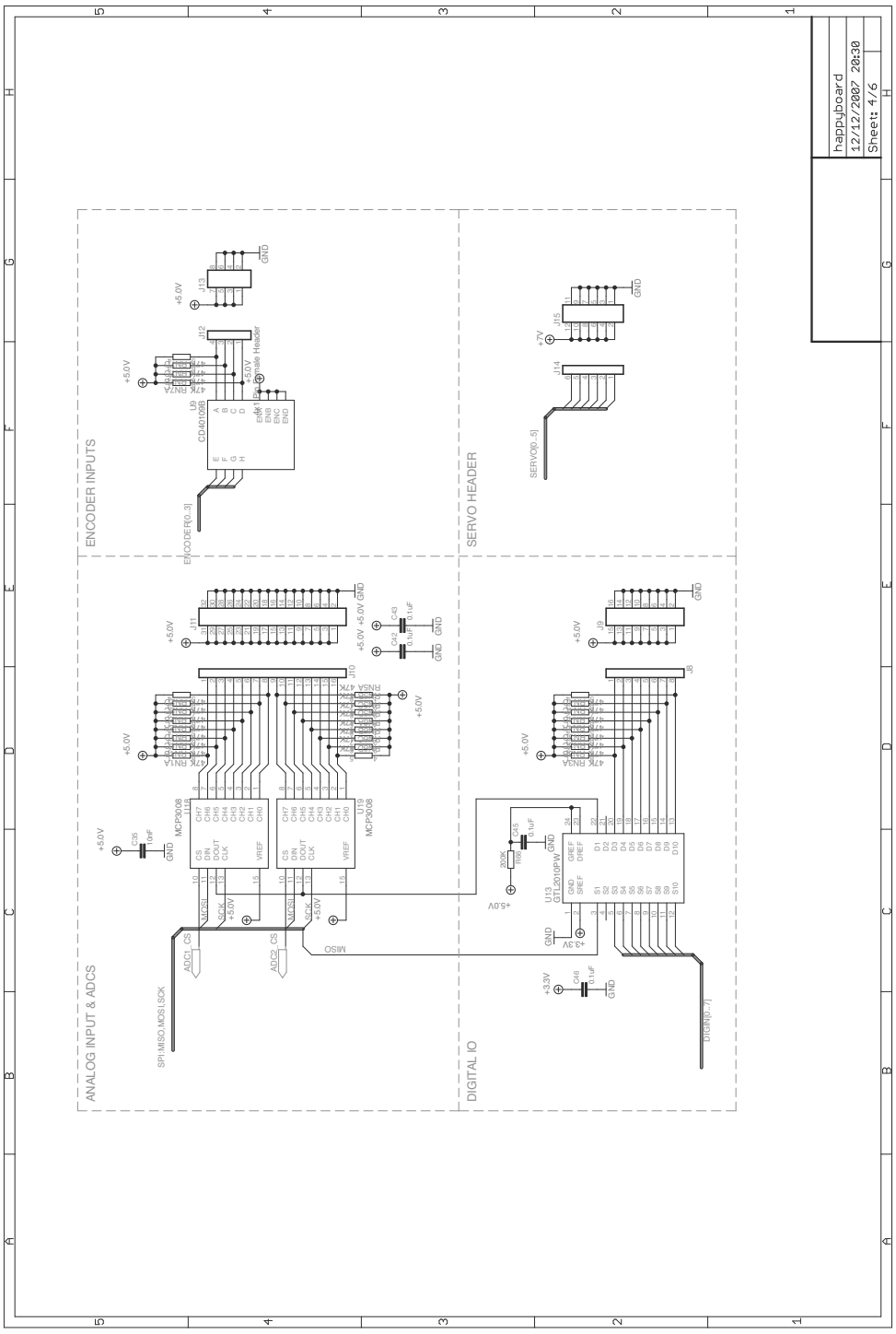
- `void init_lock ( struct lock *`*`k`*`, const char *`*`name`* `)`
  Initialize the lock *k*. This routine must be called before the lock is used.
- `void acquire ( struct lock *`*`k`* `)`
  Acquire the lock *k*. If another thread holds the lock yield until it is available, then take it. A thread can recursively acquire the same lock multiple times, but it must release once for every acquire.
- `void release ( struct lock *`*`k`* `)`
  Release the lock *k*.

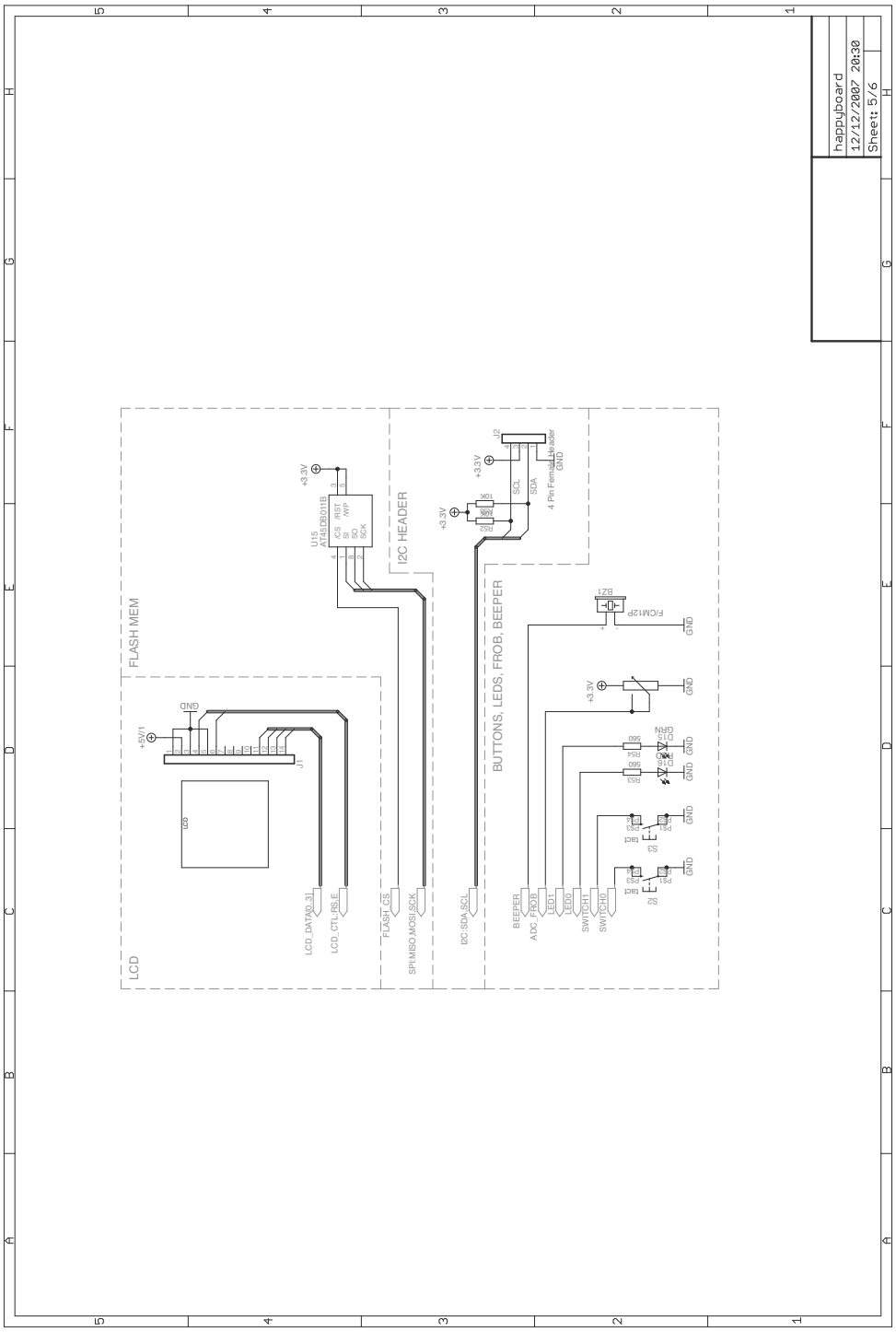
Schematics

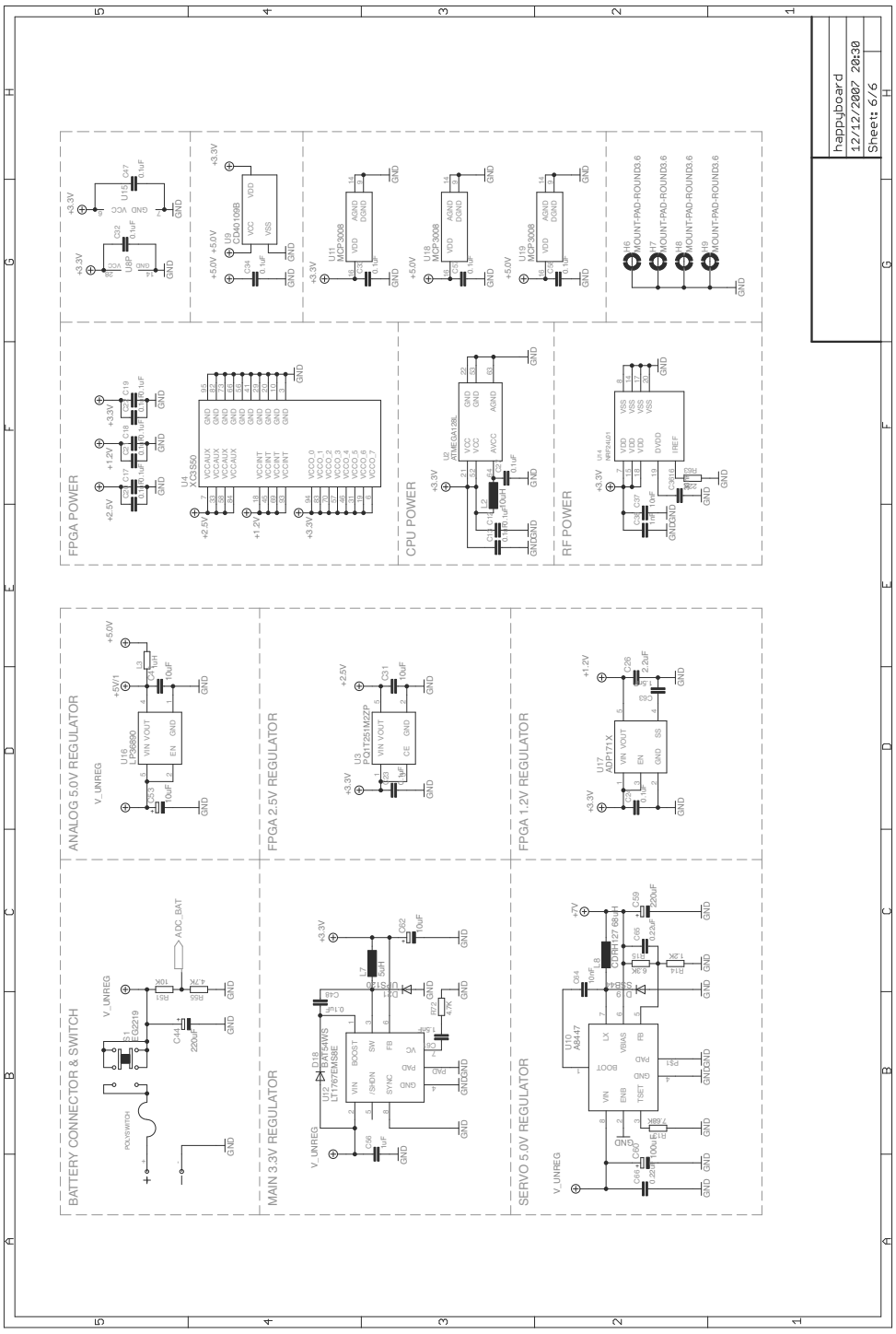

happyboard
12/12/2007 20:30
Sheet: 1/6
CPU
RAM
RESET + JTAG
RF
USB
ATMEGA128L
CY7C199
FT232RL
MINIUSB
NRF24L01
RFSMA
MCP130T
ADDRESS[0..7]
CTL_WR/RD,ALE
FPGA_CONF...
DATA_0[0..7]
ADDRESS[8..15]
SPI_MISO,MOSI,SCK
I2C_SDA,SCL
BEEPER
ADC1_CS
LCD_CTL,RS,E
LCD_DATA[0..3]
LED1
LED0
SWITCH0
SWITCH1
RF_CS
ADC_CS
JTAG:TDI,TDO,TMS,TCK
RF_CE
RAM_CS
ADC_FROB
ADC2_CS
RESET
+3.3V
GND
16MHz
8MHz
USB


happyboard
12/12/2007 20:30
Sheet: 2/6
FPGA
FPGA CONFIG
FPGA CLOCK
U4
XC3S50
DIGIN[0..7]
ENCODE[0..3]
CTL.A4,B4,EN4
CTL.A5,B5,EN5
CTL.A2,B2,EN2
CTL.A3,B3,EN3
CTL.A0,B0,EN0
CTL.A1,B1,EN1
SERV[0..5]
RAM_CS
DATA0
INIT_B
+3.3V
+2.5V
GND
DATA[0..7]
ADDRESS.A[8..15]
ADDRESS.A[0..7]
CTL.WR,RD,ALE
FGPA_CONF


happyboard
12/12/2007 20:30
Sheet: 3/6
MOTOR ADC
MOTORS 0&1
MOTORS 2&3
MOTORS 4&5
U11
MCP3008
SPI_MISO_MOSI_SCK
MOT_CS
+3.3V
M5_SENSE
M4_SENSE
M3_SENSE
M2_SENSE
M1_SENSE
M0_SENSE
ADC_BAT
V_UNREG
L6205N
VBOOT
GND
J5
J6
J7


ENCODER INPUTS
SERVO HEADER
ANALOG INPUT & ADCS
DIGITAL IO
happyboard
12/12/2007 20:30
Sheet: 4/6


happyboard
12/12/2007 20:30
Sheet: 5/6
FLASH MEM
I2C HEADER
BUTTONS, LEDS, FROB, BEEPER
LCD
U15
AT45DB011B
+3.3V
GND
+5V1
J1
LCD
LCD_DATA0..3
LCD_CTL_RS_E
FLASH_CS
SPI:MISO,MOSI,SCK
I2C: SDA,SCL
J2
4 Pin Female Header
SCL
SDA
R52
R53
R54
R55
BEEPER
ADC: FROB
LED1
LED0
SW1:CH1
SW1:CH0
BZ1
F/CM12P
SW1
SW2
SW3
SW4
SW5
SW6


BATTERY CONNECTOR & SWITCH
ANALOG 5.0V REGULATOR
FPGA POWER
MAIN 3.3V REGULATOR
FPGA 2.5V REGULATOR
CPU POWER
SERVO 5.0V REGULATOR
FPGA 1.2V REGULATOR
RF POWER
happyboard
12/12/2007 20:30
Sheet: 6/6