# Carnegie-Mellon University
## –Silicon Valley Campus

CyLab SensorFly
Toolchain Installation Manual

revision. 2009-10-06

## Components

The toolchain we use for development of SensorFly is comprised of:

•**The Olimex USB drivers and connectors** – to have a serial connection into the SensorFly hardware through our USB port.
•**OpenOCD** – a remote debugger ("remote" because it runs on our workstations rather than in the SensorFly board), it "talks" to the SensorFly through the serial connection using the JTAG protocol, obtains/sets the SensorFly state, and exposes this state through a tcp port via its own GDB-protocol-compliant server, so we may debug with GDB.
•**Yagarto** – a GNU toolchain distribution that targets the ARM processor, it includes the gcc compiler and the make command. It is designed to integrate with Eclipse well.
•**Eclipse CDT** - an Eclipse distribution intended for C/C++ development, it includes the GDB debugger, the launcher, etc...

## Instructions

Connecting to the SensorFly requires that one install the above software first. The entire toolchain is contained in the Olimex CD (if a little out of date). The Olimex USB drivers are Windows-only so far. The Olimex drivers have worked for us in both Windows XP and Windows 7.

The first thing to do is get the source code for the project. Use program Subversion to check out the source from the SensorFly project into a directory. You may use Tortoise SVN or any other subversion tool for this. The URL you should check out from is: http://sensorfly.googlecode.com/svn/tags/afly

Copy the lpc2xxx_armusb_tiny.cfg file into the c:\ root directory (the path to this file will become an argument to openOCD, and spaces in the path to this file introduce problems). This file has configuration data for communicating with JTAG to the SensorFly.

Once everything is installed, you will need to open Eclipse (normally it was installed in

c:\gccfd\eclipse\eclipse.exe), and follow these instructions:

1. On the “C/C++” panel at the left, select all items and right-click to select “delete” (but do not check the checkbox to also delete from the filesystem).
2. Under the “File” menu, select “Import”.
3. Under the “General” category, select “Existing projects into workspace”.
4. On the “Select root directory” section, click the “browse” button and browse to the directory where you checked out the source from the SensorFly Project.
5. Make sure the “sensorfly” project is checked in the checkbox list, and click “Finish”.
6. Find the “External Tools” button (the button has a white triangle in a green circle and has a red square on the lower right) and **click on the down arrow**.
7. Enter the “External Tools Configurations” dialog box.
8. Select the “Open OCD Tiny” configuration from the left panel.
9. Update the variables in the “Main” tab so that $project_loc resolves to the directory you checked out the source to. If you find it easier, then just replace “$project_loc” with the path to the source you checked out.
10. Change the “Arguments” panel so that the text reads “-f c:\lpc2xxx_armusb_tiny.cfg" (without the quotes) and close the dialog box. Remember you copied this file to c:\ before starting these instructions.
11. Build the Project (ctrl-b). After a successful make, you should have the file “main.out” in your project directory, if you have problems with make, open the makefile and check to see whether the paths to gcc and other tools are correct. Specifically, check the version number of the command that makes the target “main.out” because it is very sensitive to version numbers. Ignore errors that complain about target “clean”; the first time you make the project there's nothing to clean!.
12. Find the debug button (the one that has a bug in it) and **click on the down arrow**.
13. Enter the “Debug Configurations” dialog box.
14. Select the “Zylin embedded debug (Native)” item from the left pane, and click on the “New launch configuration” button on the upper part of the pane. This will open a new pane to the right for editting the settings for this new configuration.
15. In the right pane, set the name to something like “Sensorfly-debug”
16. Go to the “main” tab.
17. For setting the “Project (Optional)” textbox, click on browse and select the sensorfly project (you imported it into your workspace a few steps ago).
18. Under “C/C++ Application” click on the “Search project...” button and get to the “main.out” file in the project directory (this file got created when you built the project). If you get an empty box it means your project hasn't been built, so go back and build it. Remember if the make fails it might be that you need to check the paths to tools in the makefile for the target “main.out”.
19. Go to the “debugger” tab
20. In the “GDB debugger” click the “browse” button and navigate to C:\gccfd\yagarto\bin\arm-elf-gdb.exe ... it's the path to gdb for arm that came with yagarto.
21. About the “GDB command file” textbox, clear it so it is blank.
22. Go to the “Commands” tab.
23. Type in the following commands on the “ 'Run' commands” text box, they are copied verbatim from the “run_commands.txt” file in the source. (Leave the “ 'Initialize' commands” textbox blank) :

```
target remote localhost:3333
monitor sleep 500
monitor poll
```

```
monitor flash probe 0
monitor flash protect 0 0 26 off
monitor flash erase 0 0 10
monitor flash erase_check 0
monitor flash write_image main.bin 0x0 bin
monitor reset run
monitor sleep 500
monitor soft_reset_halt
monitor arm7_9 force_hw_bkpts enable
symbol-file main.out
thbreak main
continue
```

18. Go to the “Common” tab. Check “sensorfly” under the “show in favourite items” menu.
19. Close the dialog.
20. Connect the helicopter to the computer and make sure it is “on” (the jumper is on the switch, a power source is connected, etc...).
21. unplug the usb cable, then plug it again.
22. Go to the “Debug” view (it should be beside the “C/C++” view button on the right-hand side).
23. Make sure no processes are running on the debug view, if they are, terminate them all.
24. Find the down arrow beside the “External tools” button and click on “OpenOCD tiny”
25. You should see an openocd message in the console displayed in red text. You shouldn't see JTAG error messages, if that is the case, check that the helicopter is actually connected to the computer and turned on, make sure the power source connection to the SensorFly is a good connection, unplug/replug the SensorFly, and try running OpenOCD tiny again. If your problems persist, see appendix A – troubleshooting.
26. Make sure the “main.c” file from the project is open and in view.
27. Find the down arrow beside the “debug” button and click on “sensorfly” (you just created this configuration a few steps back). If you get a dialog box warning that there were errors in the project, click ignore and launch (the first time you run make, make “clean” runs but there's nothing to delete so it reports problems, ignore make problems related to target “clean” - you can get information about make problems in the “problems” tab on the bottom panel of eclipse).
28. By default, this will flash the helicopter and will set a breakpoint in the first instruction of function “main”. Click on the “Step over” button (it is the yellow arrow that looks like an inverted “u-turn” sign). This will make you step out of the trap point. Now press the Resume button (it is in the same toolbar the “step over” button is located at, it looks like a “play” button in your stereo, equivalent to F8) to run until the next breakpoint is hit. You can set arbitrary break points in the code by double-clicking on the line number you want to halt. Only 2 “hardware breakpoints” are allowed so make sure you don't have more than 2 active break points at any time.
29. To reflash the helicopter, make your changes in the C code, save them (ctrl-S), rebuild the project (ctrl-B), and relaunch the sensorfly project.
30. The “OpenOCD tiny” external tool does not need a restart every time you flash, unless you disconnect the SensorFly, then you need to restart OpenOCD tiny once you connect it. Follow the procedure described below to perform a reflash of your code:

# Reflashing the SensorFly

- Make changes to the C code

- Build the project (make sure the build is successful).
- Make sure you have a source file of the SensorFly project open and in view.
- Turn to “Debug” view.
- Make sure there are no processes running in the Debug window. If there are any, kill them all. (Most of the time this means killing two processes: the “OpenOCD tiny” remote debugger and the GDB session that connects two it).
- Make sure the helicopter is turned on and has a solid connection to its power source.
- Unplug and then plug back the usb cable. If you are running Eclipse on a virtual machine, disconnecting and then connecting the usb from the virtual machine is enough.
- Run the “OpenOCD tiny” remote debugger by clicking on the “external tools” down-arrow button and selecting it. You should see a red message in the output window that confirms OpenOCD is running. You shouldn't see any errors.
- Launch GDB to connect to the OpenOCD server by clicking on the “debug” button's down arrow and selecting the SensorFly configuration. Wait for a few seconds while the helicopter is flashed.
- You should halt at a default trap set by default on the first instruction on main(), an arrow should be pointing to the next line of code to run. If you aren't getting this, read the appendix.
- To reflash, you may kill just the GDB process (and not the remote debugger process), change the code, rebuild, and then relaunch GDB. However each time connection is lost you have to actually go through this whole procedure from the start.

# Appendix A – troubleshooting

1.- OpenOCD

Output from a successful openOCD connection looks as follows:

```
Open On-Chip Debugger 1.0 (2008-10-04-10:00) svn:exported
$URL: http://svn.berlios.de/svnroot/repos/openocd/trunk/src/openocd.c $
Info:    options.c:50 configuration_output_handler(): jtag_speed: 3, 3
Info:    options.c:50 configuration_output_handler(): Open On-Chip Debugger 1.0
(2008-10-04-10:00) svn:exported
Info:    jtag.c:1389 jtag_examine_chain(): JTAG device found: 0x4f1f0f0f
(Manufacturer: 0x787, Part: 0xf1f0, Version: 0x4)
Info:    jtag.c:1389 jtag_examine_chain(): JTAG device found: 0x4f1f0f0f
(Manufacturer: 0x787, Part: 0xf1f0, Version: 0x4)
```

Also you might get only the first line, depending on the OpenOCD version. If you get errors that complain about note being able to create a serial JTAG connection, go through this checklist:

- Try restarting openOCD tiny

- If you still get errors, disconnect the usb, then connect it again, and restart openOCD tiny
- Make sure the SensorFly has the jumper that turns it on.
- Downgrade openOCD to version 1.888
- Make sure the battery is providing more than 3.5 volts.
- If your power source is not the battery, make sure it is providing between 3.5 and 4.1 volts, and sufficient current. Sufficient current varies from values like 0.18 amps when the rotors aren't operating to like 2.8 amps when they are running at high duty cycles.

**Intermittent problems**

We encountered some situations where we could connect successfully to the SensorFly when the rotors weren't turning, but as soon as an instruction turned the motors on we lost the connection. Sometimes the problems showed up halfway through the reflashing process.

- Half of the time this was due to the power supply not providing **sufficient amperage** for the device until we turned on the rotors, at that point there wasn't enough power keep the cpu running. We regained the connection after we increased the current limit in the power source.
- The other half of the time the culprit was a **flaky connection** to the power source or to the serial cable that was ok until vibration knocked it off, and once power has been interrupted to the SensorFly, you have to go through the whole “Reflashing the SensorFly” procedure from the start. To solve this problem we added a soft material to damp vibrations at the start.

Try running with batteries if all else fails. Normally, freshly recharged batteries will give you a solid connection to the SensorFly and a sufficient amount of current, so you can at least rule out these two problems.

**Subsystems**

If you are interfacing with compass / radio, make sure your SensorFly model you are working with actually has the compass or radio chip installed. Some connections between the hardware of the sensorfly are done with $I^2C$. If the subsystem is not present, no response will be read from the channel, so your C function that is doing the call might hang if it waits for a response indefinately on the $I^2C$ channel and the chip is simply not installed.