# Introducing GP-22050

**Multi-function USB 2.0 instrument for embedded system debug & test**

*GP-22050 is Byte Paradigm advanced multi-function USB 2.0 instrument for testing and debugging embedded systems. Used as PC-controlled **pattern generator, logic analyzer, SPI and I²C host adapter,** GP-22050 has found its place on the desk of many embedded hardware and software engineers.*

*It is appreciated for its high versatility, the many software control interfaces available, its performance and compactness.*

*This paper gives an overview on the GP-22050 unique features for embedded system debug and test.*

## Testing & debugging embedded systems at functional level is all about generating stimulus and observing responses.

### Many Embedded Systems types

The Wikipedia Encyclopedia defines an **'embedded system'** as *"/…/ a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts."*

There is a complete range of embedded systems, depending on the technologies chosen to implement them. At one end of the embedded system space, there are the **processor-centric** systems. They are mainly composed of an embedded microcontroller and its peripherals. The system's functionality is almost fully **software-defined**. At the other end of this space, there is the **ASIC**, with functionality completely enclosed in one single chip and fully **hardware-defined.**

But these are not the most common (or realistic?) systems, as most embedded systems involve a **combination** of microcontrollers, peripherals, memories, communication ports, FPGA, ASIC, system-on-chip (SoC), DSP, and so on...

All of this makes of designing an embedded system a very exciting technology job that involves a mix of skills at software, hardware, mechanical, analog and digital levels.

## Testing and debugging is seen as a critical task

Many studies conducted recently – among others, a survey conducted by Byte Paradigm during the Embedded System Conference in San Jose in 2008 – designate testing and debugging as one of the most time-consuming tasks in the flow of designing an embedded system. This should not come as a surprise, as this is the direct consequence of the increasing complexity of embedded systems. Hence, testing and debugging – *that is, ensuring that the system being designed has no flaws and is functionally compliant to its specification* – has become one of the most critical tasks of embedded system design.

Testing and debugging involve many techniques: simulation, emulation, hardware acceleration, prototyping and so on. In this paper, we focus on testing and debugging an embedded system directly on hardware. This hardware can be a prototyping board with FPGA, a prototype chip received from the fab, a test board or a set of electronic boards with general computing resources used to emulate the final system, or the final system itself.

*Using a prototype during the design process is interesting because you don't have to wait until the full system is available to perform early validation on a real hardware.*

## The challenge of testing and debugging an incomplete design.

When an embedded system assembles IP, code running on a microcontroller, digital processing implemented in FPGA, ASIC, and many other 'pieces', it is likely that:

- it is the work of a whole team of engineers;
- the functional modules that compose the system are developed concurrently to speed up the planning
- not all modules are designed and available at the same time.

A very common approach consists in gradually testing and debugging each module independently prior to assemble them as a complete system. Being able to validate each module on a hardware prototype in addition to other approaches (simulation …) certainly improves the quality of the testing and debugging.

However, going on prototype for a limited part of the design poses the following problems:

- ***How to generate the inputs necessary to put my functional module under test?***
- ***How to observe the responses of the functional module under test?***

**Stimulating** and **observing** are really the 2 recurrent issues when testing and debugging an incomplete system.

In extreme cases, an engineer may end up designing a whole new separate embedded system to put another one under test, losing the benefit of going to a prototype for testing and debugging to speed up design and being more productive in the validation process.

Table 1 gives an overview of what can be used and/or implemented to stimulate and observe a prototype under test (not extensive).

## Efficient testing and debugging requires a mix of techniques

What to choose among the available tools and methodologies for stimulating and observing an embedded system? The answer is simple: mix them all!

Any embedded system is a different mix of techniques; why would this be different for what you use for testing and debugging?

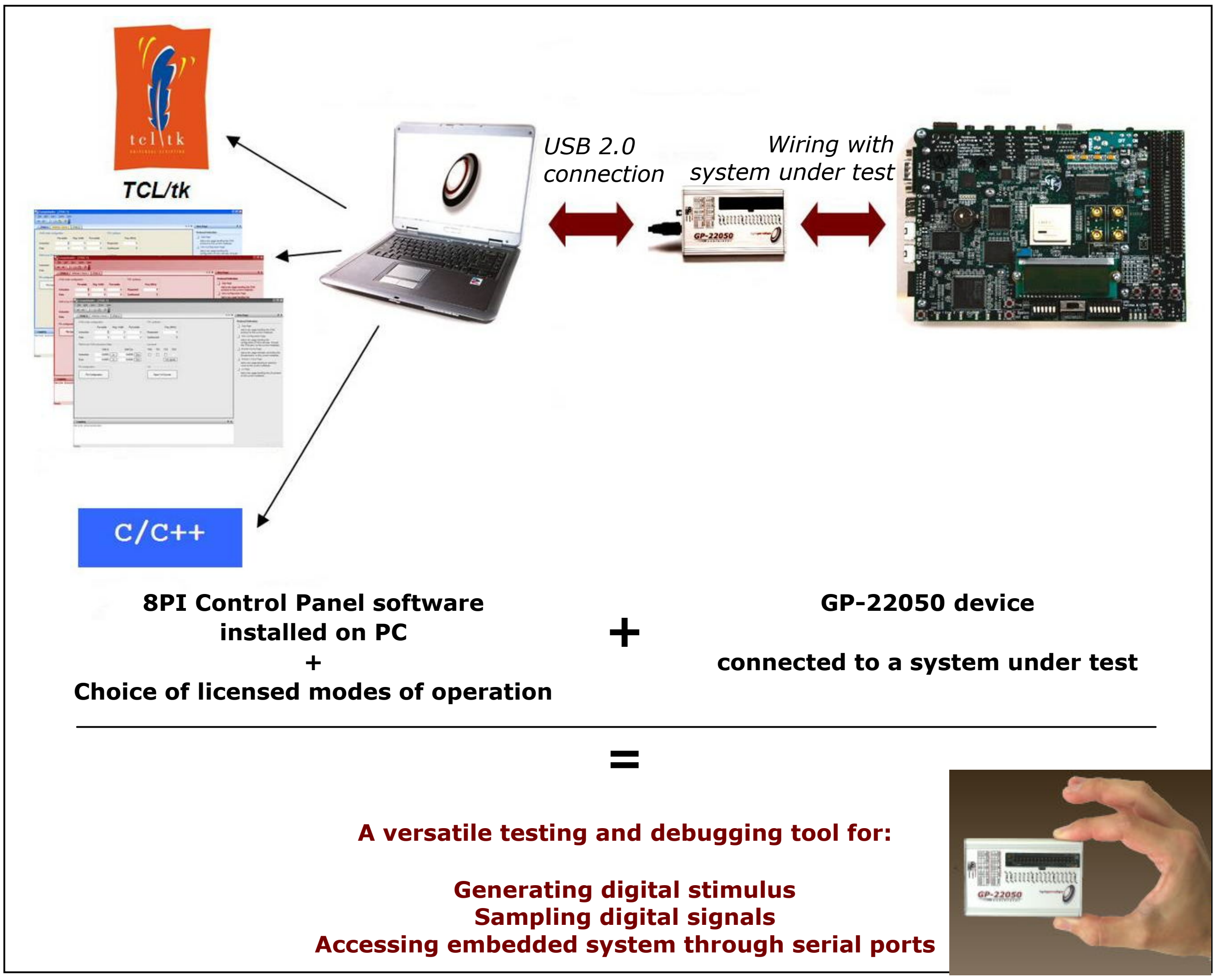
**Table 1 : Prototype stimulation / observation techniques**

| Resource type | Stimulation | Observation |
|---|---|---|
| **Embedded microcontroller** | Generate specific test code and the µC peripherals to generate stimulus for all the system.<br>Program system registers and memories. | Use the µC to read back memories and registers where test results are stored |
| **Oscilloscope** | n.a. | Probe signals |
| **Logic Analyzer** | n.a. | Sample digital signals – requires a debug connector. |
| **JTAG port** | Bring specific values on the I/Os of some devices through the boundary scan network. Can be uneasy for real-time stimuli.<br>Program devices, fill memories with specific content. | With a software emulator, allows accessing an embedded processor trace data.<br><br>With tools like embedded LA, allows collecting trace data out of FPGA, previously stored in memories (not real-time). |
| **System ports (Ethernet, PCI, …)** | Allows stimulation through the same functional ports as the final system. Requires the system ports to be designed and available. | Allows collecting trace data out of the system under test through its normal functional ports. Requires the system ports to be designed and available. |
| **Digital Pattern Generator** | Allows generating arbitrary digital stimulus at the functional inputs of the system under test. Requires the proper test and debug connector. | n.a. |
| **Protocol host adapter** | Specific to a given protocol (e.g.: SPI, I²C), allow controlling a communication port from PC (write/read). | |
| **Protocol analyzer** | n.a. | Allows analysis of the traffic on a given communication port to check compliance to a given protocol and/or physical signaling. |
| **Port monitor** | n.a. | Collects and records traffic from a given communication port of the system under test. |
| **Waveform generator** | Allows generating specific analog signal at the inputs of the system under test. | n.a. |

## GP-22050 - a 'Swiss Army Knife' for test and debug in one compact USB 2.0 device.

### One device, multiple modes of operation with a perpetual licensing model

**Figure 1 : GP-22050 principle**


TCL/tk
C/C++
USB 2.0 connection
Wiring with system under test
8PI Control Panel software installed on PC
+
Choice of licensed modes of operation
+
GP-22050 device
connected to a system under test
=
A versatile testing and debugging tool for:
Generating digital stimulus
Sampling digital signals
Accessing embedded system through serial ports

**GP-22050 targets manual bench testing and debugging on prototype system during design,** specifically on digital interfaces. Because testing and debugging scenarios involve generating digital input stimuli AND collecting digital data, GP-22050 is basically both a logic generator AND a logic analyzer.

**GP-22050** is based on **modes of operations.** A mode of operation is a coherent set of functionalities that turns the GP-22050 into a given type of device, for a given type of test and debug task.

**Each mode of operation** defines a set of:

- Configuration settings for the GP-22050, such as:
  - the number of channels used
  - the clock frequency
  - the trigger on which the device must start operating
- Operation functions, such as:
  - Starting to generate samples
  - Communicating with a given slave in SPI mode
  - Starting to acquire samples.

**8PI Control Panel** software allows controlling the GP-22050 device in all modes of operation. ***It is impossible to control the GP-22050 without installing 8PI Control Panel software.***

8PI Control Panel software manages the GP-22050 USB 2.0 connection with the PC through a special agent application called 'Smart Router'. The 'Smart Router' must always run to control the GP-22050 device. This application is the 'point of connection' of the GP-22050 through USB on one side, and each instance of the available modes of operation. If it does not run already, it is called automatically when opening an instance of any mode of operation.

Some modes of operation are provided for free when purchasing the GP-22050. Some others require purchasing an additional license. Table 2 gives an overview of the modes of operation available with GP-22050 (Q1-Q2 2009 data – please go to www.byteparadigm.com for an up-to-date information).

**Table 2: Modes of operation available for GP-22050**

| Mode of operation | Description | License information |
|---|---|---|
| **ADWG** | Turns the GP-22050 device into a **digital pattern generator**.<br><br>*Digital samples generation on up to 16 bits, max. 50 MHz – **Generation only**.* | **Available for free with the GP-22050 device.** |
| **Analyzer** | Turns the GP-22050 device into a **logic analyzer**.<br><br>*Up to 16 bits parallel sampling at max. 50 MHz – **Sampling only**.* | |
| **JTAG** | Allows using GP-22050 to access a **JTAG port**.<br><br>*Allows toggling the JTAG port signals (TDI, TMS, and TCK) and collecting TDO. Support for SVF files.* | |
| **SPI** | Turns the GP-22050 into: **SPI host adapter** (SPI Master) and **SPI analyzer**.<br><br>*Allows playing the role of a serial peripheral interface master from PC.*<br>*Allows sampling a SPI interconnect and decoding the SPI traffic (logic analyzer with SPI protocol decoding support).* | **Requires additional license.** |
| **I2C** | Turns the GP-22050 into an **I²C host adapter** (I²C Master) and **I²C analyzer**.<br><br>*Allows playing the role of an I²C master from PC.*<br>*Allows sampling an I²C network and decoding the I²C traffic (logic analyzer with I²C protocol decoding support).* | **Requires additional license.** |

## Multiple interfaces

An instance of a given mode of operation may use one of the provided following **user interfaces:**

- MS-Windows® graphical user interface;
- TCL/tk scripting interface;
- C/C++ function calls from provided DLL.

All 3 types of user interface are provided with each mode of operation licensed for the 8PI Control Panel.

## Rough performance

Each mode of operation will make the most of the GP-22050 hardware resources.
GP-22050 rough performances are summarized in Table 3.

**Table 3 : GP-22050 rough performance characteristics**

| Characteristic | Value | Comment |
|---|---|---|
| **Max. number of digital channels** | **16** | *Depending on the mode of operation, from 2 to 16 digital channels will be used.* |
| **Max. frequency on connector** | **50 MHz** | *= Maximum pattern rate in ADWG mode;<br>= Maximum sampling frequency in logic analyzer mode;<br>= Maximum data rate in SPI mode;<br>…* |
| **Embedded memory** | **16 kByte** | *Memory buffer in the GP-22050 device.* |
| **Max. throughput at connector** | **100 MByte/s** | *Equivalent to 16 channels toggling at 50 MHz.* |
| **Max. sustained throughput** | **11 MByte/s** | *When data quantities exceed the embedded buffer, the data is flowed through the USB connection. This figure shows the maximum rate sustainable by the GP-22050 with 8PI Control Panel software over the USB connection.* |

## The advantage of extended PC controllability

GP-22050 finds its place in **test labs** and on the **design desk** of many hardware and software engineers. Its natural companion is the ubiquitous PC where the designer elaborates algorithms, builds system architecture with CAD tools, types application code and hardware code (VHDL, Verilog, systemC), simulates the system at algorithmic, RTL, gate-level levels.

GP-22050 is controllable from any software environment that allows calling simple C functions. It can also be controlled on base of simple configuration and data files (text or binary) or with simple buttons and controls of the graphical user interface.

This extended PC controllability presents many advantages:

- Access through C/C++ DLL allows building custom control interfaces that fit specific testing and debugging needs, enabling direct access to essential functions and personalized use;
- You don't need to any other environment than the one you are used to. C/C++ functions can be called from most environments, from Visual Basic to LabView®.
- PC controllability ensures a real continuity between the design and simulation environment and the test environment on 'real hardware'. This is especially valuable when tests are conducted all through the design process.
- GP-22050 may be an essential part of a more complete test and debug solution.

*A very common example of using GP-22050 as a complement to an existing environment consists in installing the 8PI Control Panel software on a logic analyzer running with MS Windows and connecting the GP-22050 to the logic analyzer USB port. This builds up a **complete stimulus-and-response solution**, with the GP-22050 generating digital patterns to a board and the mainframe logic analyzer sampling the system's responses (Figure 2).*

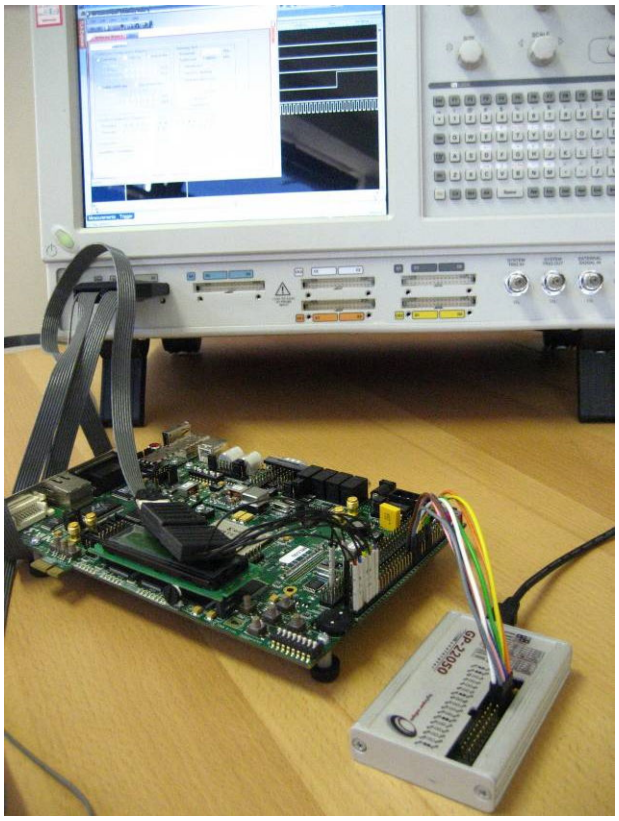


**Figure 2 : GP-22050 running with a logic analyzer**

## GP-22050 : a convenient toolbox for many testing and debugging tasks.

Generating a clock, sending samples to a DAC, accessing a system through a serial interface, checking activity on a system's I/Os… as many very common tasks required when testing and debugging a digital embedded system.

Surprisingly, it is not uncommon to spend days (if not weeks) trying to do this conveniently during design – not to mention that high-end lab equipment are not always available for common tasks.

***GP-22050 offers a personal multi-purpose solution that complements the usual test and debug environment and eases validation on prototype. As USB-powered device, it just requires plugging the USB cable, connecting a few wires and starting the 8PI Control Panel software.***

## Getting started with GP-22050

**www.ByteParadigm.com** is the best place to start with for the GP-22050. Here are very useful links:

- **GP-22050 page** is the starting point for all GP-22050-related information.
- **GP-22050 data sheet** provides extensive information over the GP-22050 device, especially the maximum ratings and hardware characteristics.
- **8PI Control Panel user's guide** is where you'll find features are offered by each mode of operation.
- **Understanding GP-22050 key performance figures** technical note focuses on explaining what is behind the GP-22050 rough figures.
- **C libraries user's guides**, found on our **general documentation page** provide an extensive list of the functions accessible through C function calls.
- **TCL/tk libraries user's guides** and all technical documentation are accessible from our **general documentation page**.
- Any complementary technical information can be requested from: support@byteparadigm.com