# The main feature of the proposed system

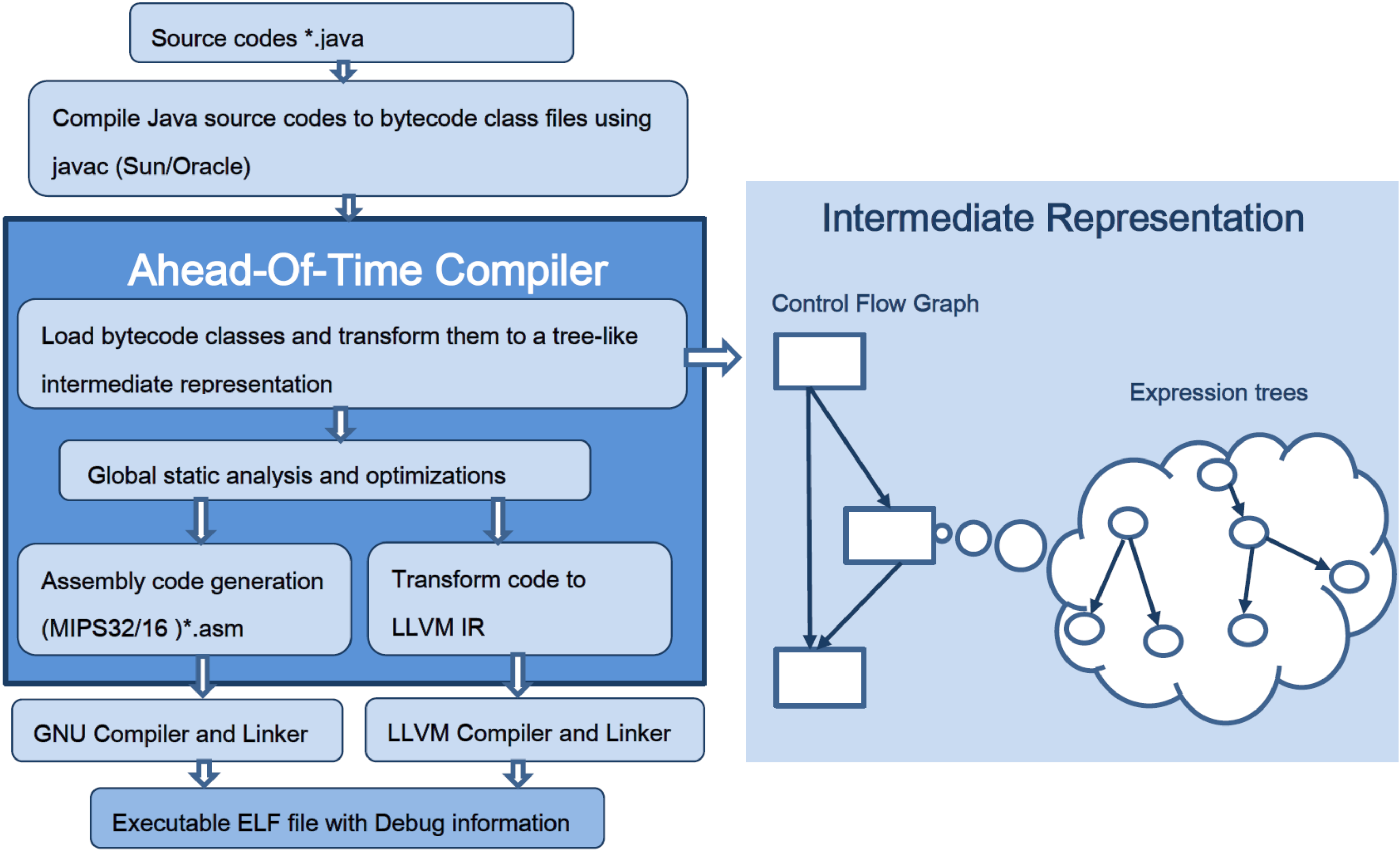
**Easy and effective programming in Java with speed and fast response inherent to C**

# System description

The system consists of the following main parts

- Fully static compiler
- The compact and efficient runtime
- The GUI design can be extended with industry standard frameworks and APIs

# Ahead-Of-Time compiler structure



Source codes *.java

Compile Java source codes to bytecode class files using javac (Sun/Oracle)

## Ahead-Of-Time Compiler

Load bytecode classes and transform them to a tree-like intermediate representation

Global static analysis and optimizations

Assembly code generation (MIPS32/16 )*.asm

Transform code to LLVM IR

GNU Compiler and Linker

LLVM Compiler and Linker

Executable ELF file with Debug information

## Intermediate Representation

Control Flow Graph

Expression trees

# Fully static compiler

- Translation of Java programs directly to the MIPS assembler
- Seamless integration of Java programs with C programs in the same project
- Global static analysis in the compiler
- Fully static garbage collection

# Translation of Java programs directly to the MIPS assembler

*The system contains the original compiler, which uses *.class files, generated by javac as input.*

*The compiler generates intermediate file in MIPS assembly language(MIPS32 and MIPS16).*

*The compiler also can generate code in the form of intermediate representation for LLVM, which then can be translated into an efficient assembly code for MIPS.*

# Seamless integration of Java programs with C programs in the same project

*- Methods of the Java classes have a direct correspondence to the C functions, no intermediate layer like JNI;*

*- The project could be a mix of Java code and C code, there is Java subtree and C subtree in the main project;*

*- Building the project will be done by simple pressing a key, as is common in C projects;*

*- Debugger is easy to use, it shows breakpoints and debug info in Java and C sources;*

*- MPLAB X could be easily extended with presented system.*

# Global static analysis in the compiler

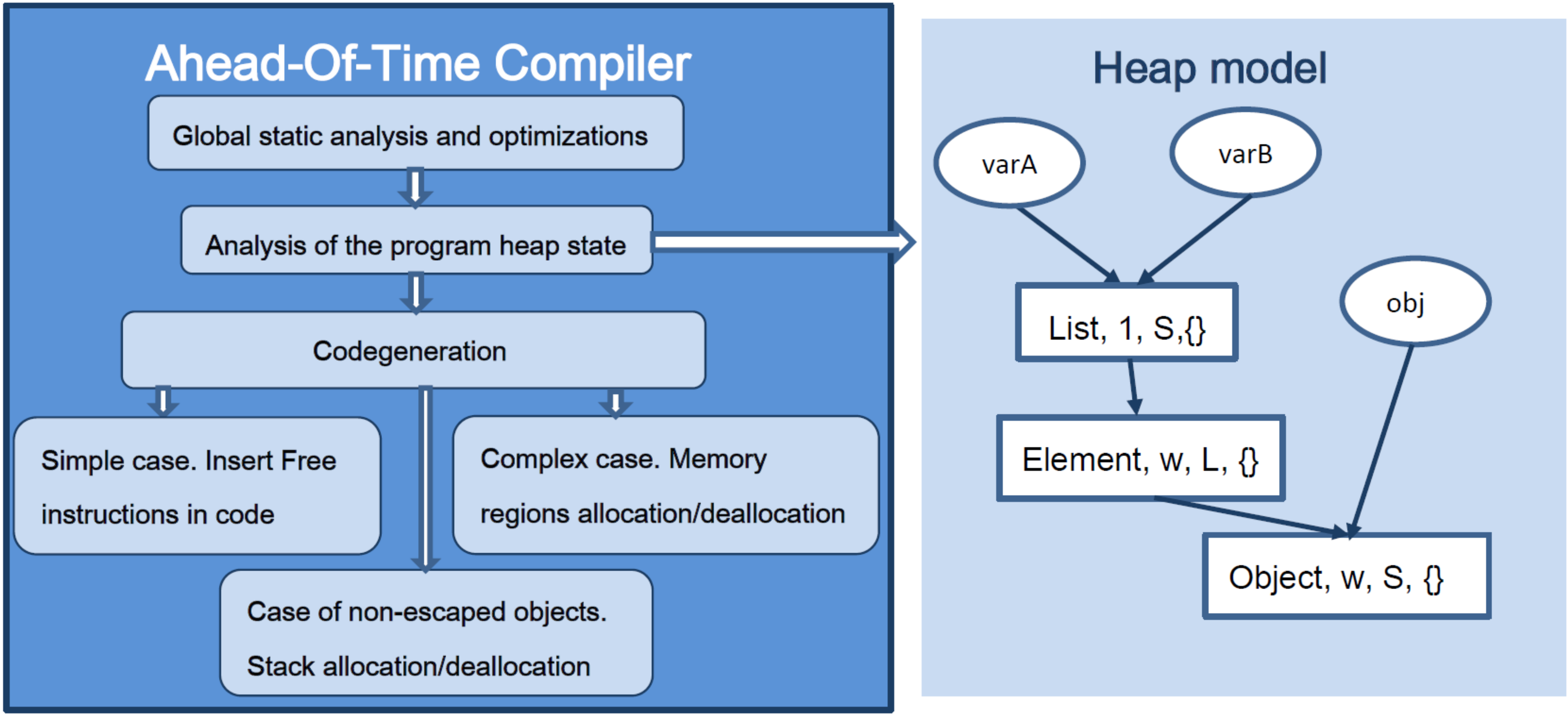
*The compiler performs  a series of global static analysis of the code. There are some analysis results:*

- *Removing unused virtual method calls;*
- *Eliminating redundant monitors in synchronized methods, etc;*

*Such code transformations improves performance and reduces code size, which is very important for embedded systems.*

# Fully static garbage collection

*The garbage collection is a very important problem in the embedded systems, traditional methods creates too many non-determinism in program behavior. This problem is solved in presented system, garbage collection is made entirely statically.*



Ahead-Of-Time Compiler

Global static analysis and optimizations

Analysis of the program heap state

Codegeneration

Simple case. Insert Free instructions in code

Complex case. Memory regions allocation/deallocation

Case of non-escaped objects. Stack allocation/deallocation

Heap model

# The compact and efficient runtime system

- Ultra compact runtime core, which contains Java basic functionality ( memory allocation , multitasking management , exceptions , …)

- Library of Java classes, for the various PIC32 peripheral devices

# Compact runtime core

*The kernel size is less then 1 Kb minimum and is about 8-14 Kb typical of flash memory. All basic functions are implemented directly over CPU without any intermediate level. For example:*

- *Multitasking is implemented as native methods java.lang.Thread;*
- *Synchronization is a direct implementation of the function enter to monitor or exit;*
- *Notification waiting is implemented as native methods java.lang.Object.wait/notify.*

*So, typical kernel size is about few Kb and this is not overhead cased by using Java, because in fact, there is a special equivalent of the built-in RTOS is provided.*

# Library of Java classes, for the abstraction of various PIC32 peripheral devices

*System provides an easy work with peripheral devices. Library of Java classes contains classes for different cases:*

- *Various Java classes for standard PIC32 peripherals(USART, SPI, I2C, ADC, Timer, Inputs/Outputs, Interrupted Inputs, PWM, Input Capture);*
- *Special classes for other devices(Matrix Keyboard, Lightweight Timer), including devices from the robotics (Servo Motor, Wheel Encoder, Ultrasonic Range Meter);*
- *The work with interrupts is comfortable, low-level part is hidden form user by default and a practical interrupt handling uses abstraction like DPC or interrupt listeners;*

*All these features makes easy to create devices, for example based on Arduino, but is much more convenient to create actual complex projects.*

# The GUI design can be extended with industry standard frameworks and APIs

- Creating a GUI with common IDE, depending on the class of devices - J2ME or Android

- Compiling a GUI declarative structures in MIPS assembler and linking with the rest part of the project

- Using of specially adapted graphical runtime library

# System Use-Cases

| MICROCHIP<br>PIC32<br>Microcontroller<br>Families | J2ME | | ANDROID | |
|---|---|---|---|---|
| | Without Display | Low Resolution Display | Low Resolution Display | High Resolution Display |
| PIC32MX<br>Families 1,2 | ▓ | ▒ | ⬜ | ⬜ |
| PIC32MX<br>Families 3,4 | ▓ | ▓ | ░ | ⬜ |
| PIC32MX<br>Families 5,6,7 | ▓ | ▓ | ▒ | ░ |
| PIC32MZ | ▓ | ▓ | ▓ | ▒ |
| PIC32MZ<br>Ext. SDRAM | ▓ | ▓ | ▓ | ▓ |

| ⬜ | ░ | ▒ | ▓ |
|---|---|---|---|
| Not suitable | Less suitable | Suitable | Most suitable |

# Meteo station example

The application example is a simple meteo station. The meteo application reads current temperature and time shows temperature on display and stores data in EEPROM.

There are two boards by Digilent - chipKIT Uno32 and chipKIT Basic I/O Shield. The Uno32 board has PIC32MX320F128 with 128K of flash program memory and 16K of SRAM. The Basic I/O Shield provides EEPROM, RTC and OLED 128*32 display.

The application consists of several parts:

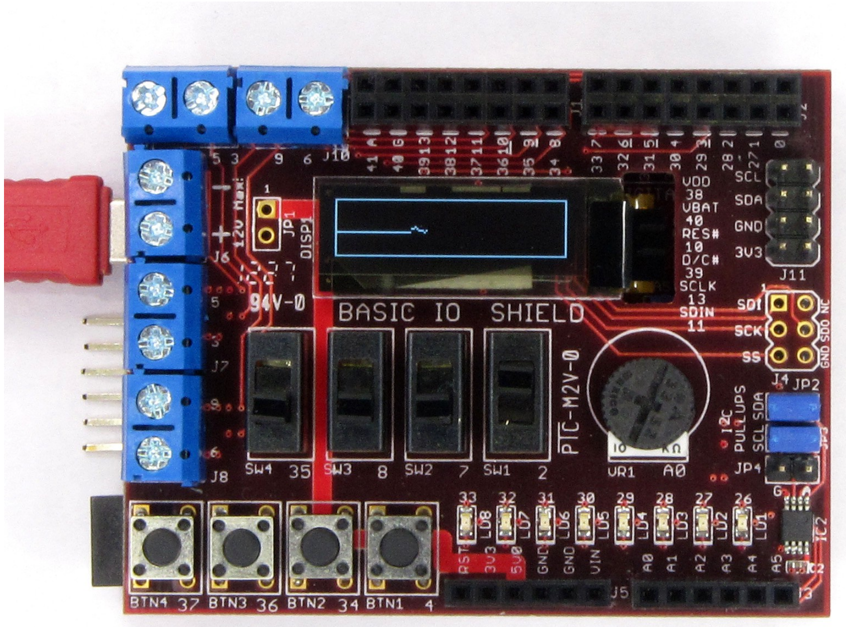
- Screen;
- Key handler;
- Temperature archive;
- UART terminal.

The application uses multitasking with several threads.

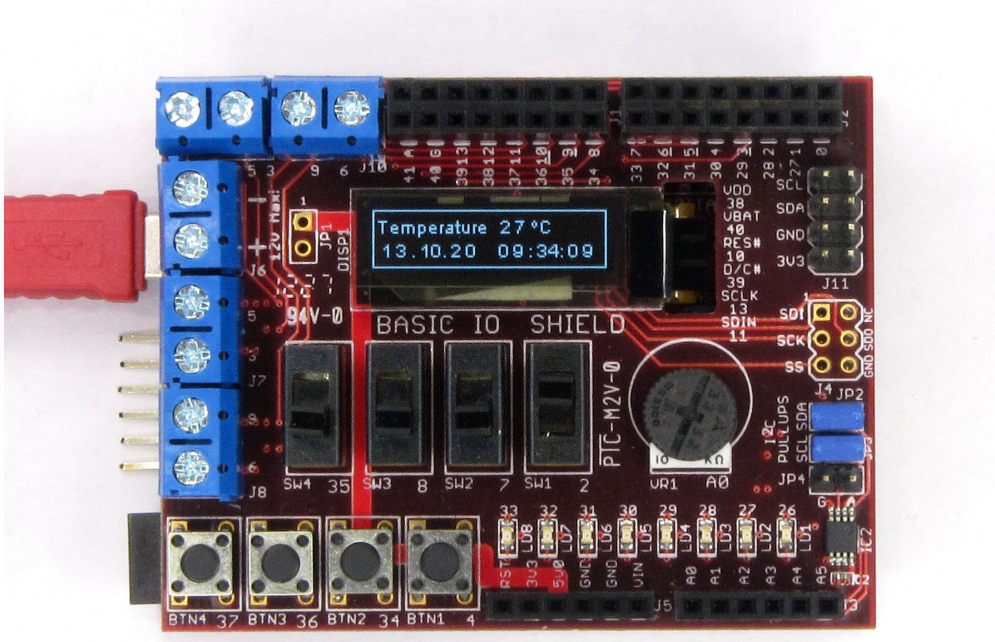
Application memory using:

Used heap size — 3550(SRAM consumption for dynamic heap objects)

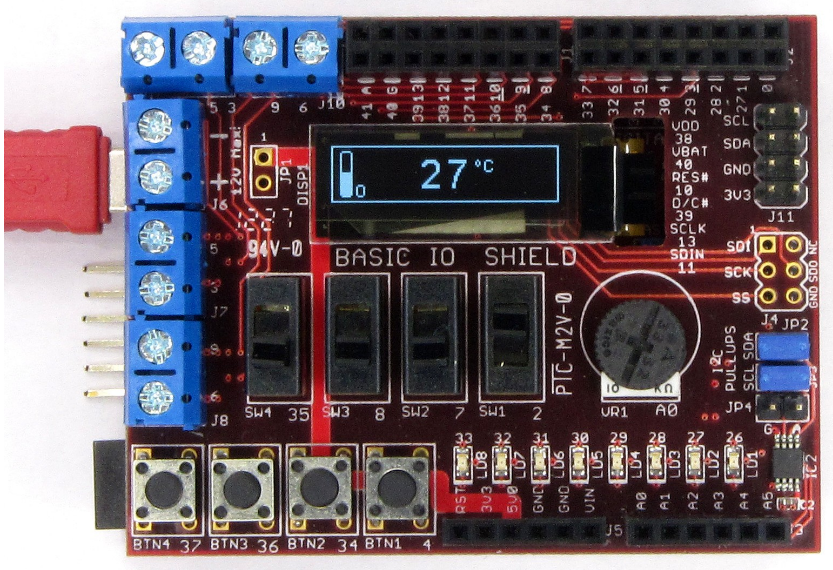
# Meteo station example



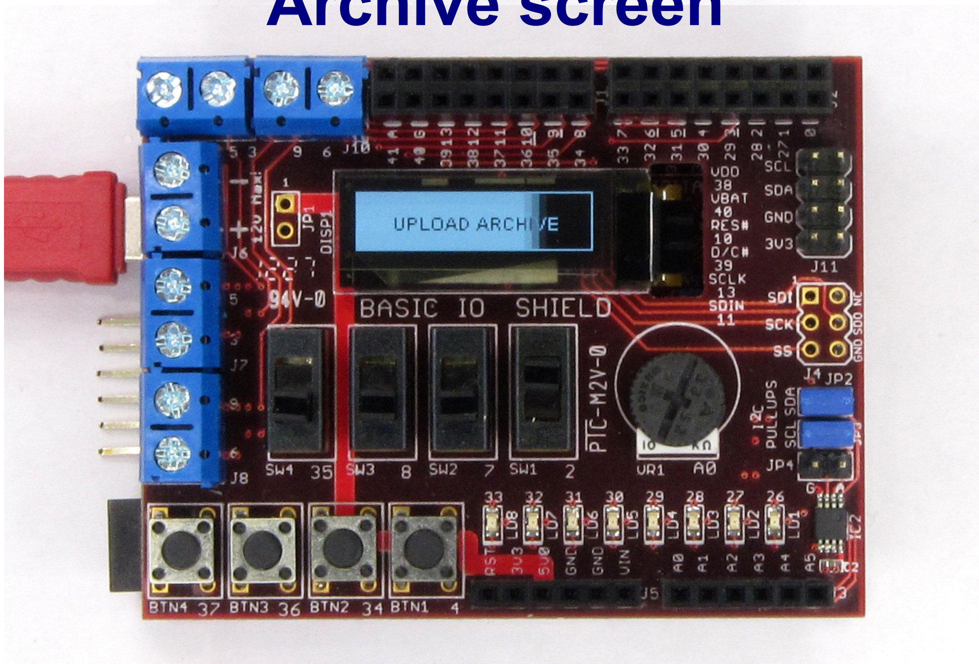
Graph screen



Archive screen



Main screen



Archive uploading by UART