Getting Started with SDCC for the STC89C52RC: Command Line and Geany

Overview

This tutorial walks you through installing SDCC, creating your first 8051 project, compiling firmware, and flashing an **STC89C52RC** board – both from the raw command line and from inside the Geany editor. It is written for beginners on Windows, and assumes MiKTeX/WinEdt only for editing this document itself; no LaTeX is needed for the firmware workflow.

Unlike many older 8051 tutorials, the STC89C52RC does *not* use a SPI/ISP programmer such as `avrdude` with a USBasp. It has a UART bootloader built into the chip, and is programmed over a plain serial connection using STC's own ISP tool. That distinction matters, and it shapes the whole flashing section below.

Installing SDCC

SDCC (Small Device C Compiler) is a free, open-source compiler for 8051 microcontrollers.

Download and Install

Download SDCC from the official SourceForge page and run the installer. Prefer a numbered stable release (e.g. `sdcc-4.4.0-setup.exe`) over a nightly/snapshot build – snapshot builds are more likely to ship internal binaries that are out of sync with each other.

Verify Installation

Open a command prompt and type:

```
sdcc --version
```

A correct install prints something like:

```
SDCC : mcs51/... 4.4.0 #<revision> (<date>)
```

The PATH Gotcha

If you also have a separate MinGW/GCC toolchain installed for general Windows console development, its `cc1` can end up ahead of SDCC's on your **System** PATH. SDCC's preprocessor (`sdcpp.exe`) calls `cc1` by bare name, expecting to find the SDCC-patched

copy sitting next to it in `SDCC\bin`. If a plain GCC `cc1` is found first instead, you will see errors like:

```
cc1: error: unrecognized command-line option '--obj-ext=.rel'
-:0: warning 190: ISO C forbids an empty translation unit
subprocess error 1
```

even though `sdcc --version` works perfectly fine on its own.

Windows builds the effective PATH as **System PATH first, then User PATH** – so even if your User PATH lists SDCC first, a System PATH entry pointing at a GCC `libexec` folder (e.g. `C:\Users\you\gcc\libexec\gcc\x86_64-w64-mingw32\<ver>`) will win the search first and shadow SDCC's `cc1`.

Fix: open *System* environment variables (not just User), and either remove that GCC `libexec` entry from PATH entirely (GCC doesn't need it exposed to work normally), or move `C:\Program Files\SDCC\bin` above it. Confirm with:

```
where cc1
```

SDCC's copy should now be listed first. Open a *new* command prompt (and restart Geany, if it was already running) before testing again – both cache the environment at launch and won't see a PATH change until restarted.

Creating Your First Project

Create a new folder for your project, for example:

```
LedBlink
```

Inside this folder, create a file named `LedBlink.c` and add the following code:

```
#include <8052.h>

void delay(void);

void main(void) {
    while(1) {
        P1 = 0xFF; // Turn on all LEDs on Port 1
        delay();
        P1 = 0x00; // Turn off LEDs
        delay();
    }
}

void delay(void) {
    int i, j;
    for(i=0; i<0xFF; i++)
        for(j=0; j<0xFF; j++);
}
```

This simple program blinks all LEDs connected to Port 1.

Command-Line Workflow

Compiling

Open a command prompt inside your project folder and run:

```
sdcc LedBlink.c
```

This compiles and links in one step, producing a `.ihx` (Intel HEX) and a `.map` (memory map) file.

Build Script: buildhex.bat

Rather than typing the compile command and renaming the output by hand each time, use a small wrapper script. Save this as `buildhex.bat` in your project folder:

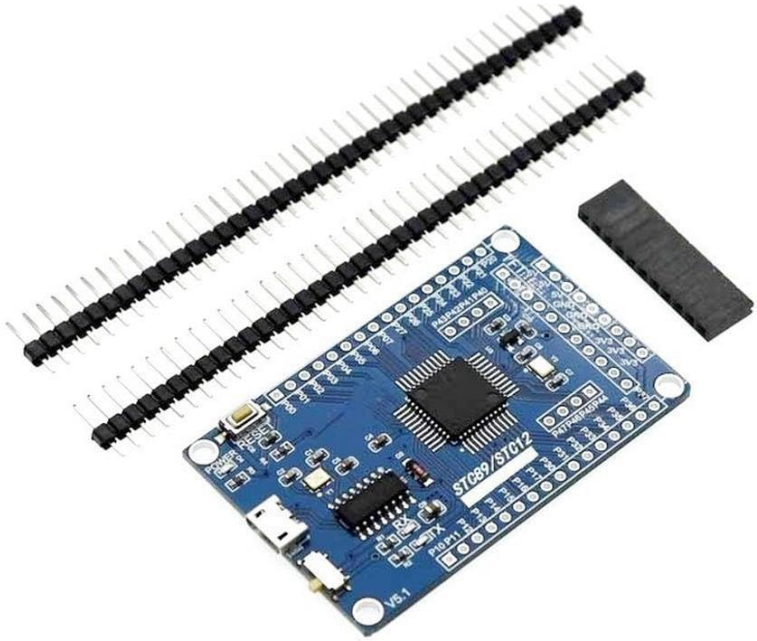
```
@echo off
REM ============================================================
REM SDCC compile wrapper to produce .hex instead of .ihx
REM Usage: buildhex.bat sourcefile.c
REM ============================================================

if "%~1"=="" (
    echo Usage: %~nx0 sourcefile.c
    exit /b 1
)

REM Compile with SDCC (adjust include paths/options as needed)
sdcc %~1
if errorlevel 1 (
    echo [ERROR] Compilation failed.
    exit /b 1
)

REM Get base filename without extension
set "basename=%~n1"

REM Check if .ihx exists
if not exist "%basename%.ihx" (
    echo [ERROR] No .ihx file found. Compilation may have failed.
    exit /b 1
)

REM Rename .ihx to .hex (overwrite if exists)
del /f /q "%basename%.hex" >nul 2>&1
rename "%basename%.ihx" "%basename%.hex"

echo [OK] HEX file generated: %basename%.hex
```

Run it with:

```
buildhex LedBlink.c
```

Note this script *renames* the `.ihx` to `.hex` rather than running it through `packihx`. A straight rename skips SDCC's address-record reformatting; STC's own ISP tool accepts the renamed file without issue, but if you ever switch to a different programmer and see address-related flashing errors, that's the first thing to revisit.

Flashing an STC89C52RC



A typical STC89/STC12 development board (v5.1), as commonly sold on Amazon and AliExpress. Note the micro-USB port (used with an onboard USB-to-serial chip), the POWER and RESET buttons, and the TX/RX indicator LEDs near the USB connector.

Connect the board to your PC over USB-to-serial (the STC89C52RC has no ISP/SPI header – it uses a UART bootloader built into the chip). On boards like the one above, the micro-USB port *is* the serial connection – no separate USB-to-serial adapter is needed, since the converter chip is already built onto the board. When Windows enumerates it, that's the COM port to use for both `PORT` in `program_hex.bat` and the port you selected in Device Manager.

Flash Script: program_hex.bat

Save this alongside `stcisp.exe` in your tools folder:

```
@echo off
REM ============================================================
REM program_hex.bat - Program an Intel HEX file to STC MCU
```

```
REM Port fixed to COM3
REM ============================================================

setlocal

set PORT=COM3
set BAUD=2400
set STCISP=%~dp0stcisp.exe

if "%~1"=="" (
    echo.
    echo Usage: program_hex.bat filename.hex
    echo.
    echo Example: program_hex.bat blink.hex
    echo program_hex.bat firmware.ihx
    echo.
    exit /b 1
)

if not exist "%~1" (
    echo Error: file not found - %~1
    exit /b 1
)

if not exist "%STCISP%" (
    echo Error: stcisp.exe not found in %~dp0
    echo Make sure this batch file is in the same folder as stcisp.exe
    exit /b 1
)

echo.
echo ========================================
echo STC ISP Programmer
echo Port : %PORT%
echo Baud : %BAUD%
echo File : %~1
echo ========================================
echo.
echo Power-cycle the MCU when you see "Waiting for MCU..."
echo.

"%STCISP%" -s %PORT% -b %BAUD% -f "%~1"

if errorlevel 1 (
    echo.
    echo Programming FAILED
    exit /b 1
) else (
    echo.
    echo Programming finished
)
```

```
endlocal
```

Adjust `PORT` to match your board's COM port (check Device Manager). The baud rate is fixed at **2400**, which has proven the most reliable rate for this setup – faster rates are supported by the chip but are more prone to framing errors on some USB-serial adapters.

Run it with:

```
program_hex LedBlink.hex
```

The Handshake

Unlike an ISP/SPI programmer, the STC bootloader only wakes up at power-on. The tool will print a waiting message and expect you to **power-cycle the board** at that point – unplug and reconnect power (or press a reset/power switch if your board has one). Once the chip's bootloader responds, you'll see a `TX: 70H` line in the output, after which programming completes on its own.

Setting Up Geany as Your IDE

Geany can run the same `sdcc`, `buildhex.bat`, and `program_hex.bat` steps from its Compile / Build / Execute buttons, once configured correctly.

Open Build → Set Build Commands

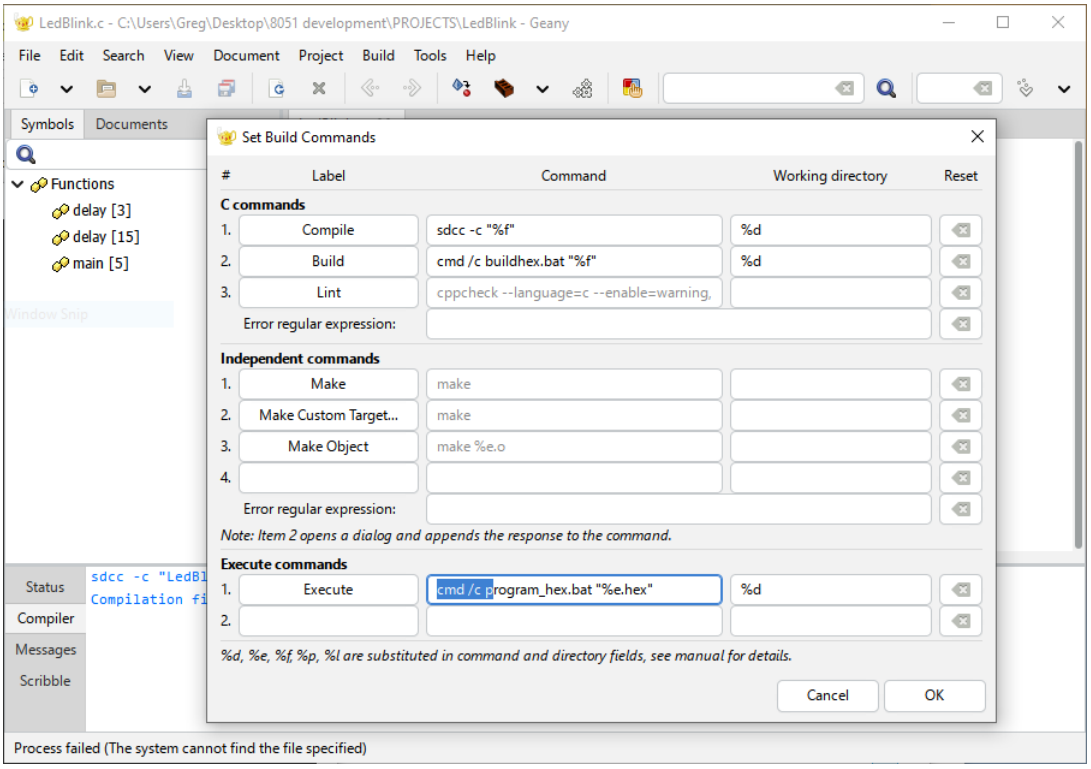
With `LedBlink.c` open as the active file (not a blank/untitled window), Geany shows filetype-specific rows for C. Set them as follows:

Row	Command	Working directory
Compile	`sdcc -c "%f"`	`%d`
Build	`cmd /c buildhex.bat "%f"`	`%d`

And under Execute commands:

Row	Command	Working directory
Execute	`cmd /c program_hex.bat "%e.hex"`	`%d`

Here `%f` is the filename with extension, `%e` is the filename without extension, and `%d` is the directory of the currently open file.



The Set Build Commands dialog with Compile, Build, and Execute rows filled in as above.

Why cmd /c Is Required

Geany launches build commands directly via Windows' process-creation API, which can only start real executables (`.exe` files) on its own – it does not know how to interpret a `.bat` script the way a command-prompt session does. Pointing Geany straight at `buildhex.bat` or `program_hex.bat` without a shell in front of it produces:

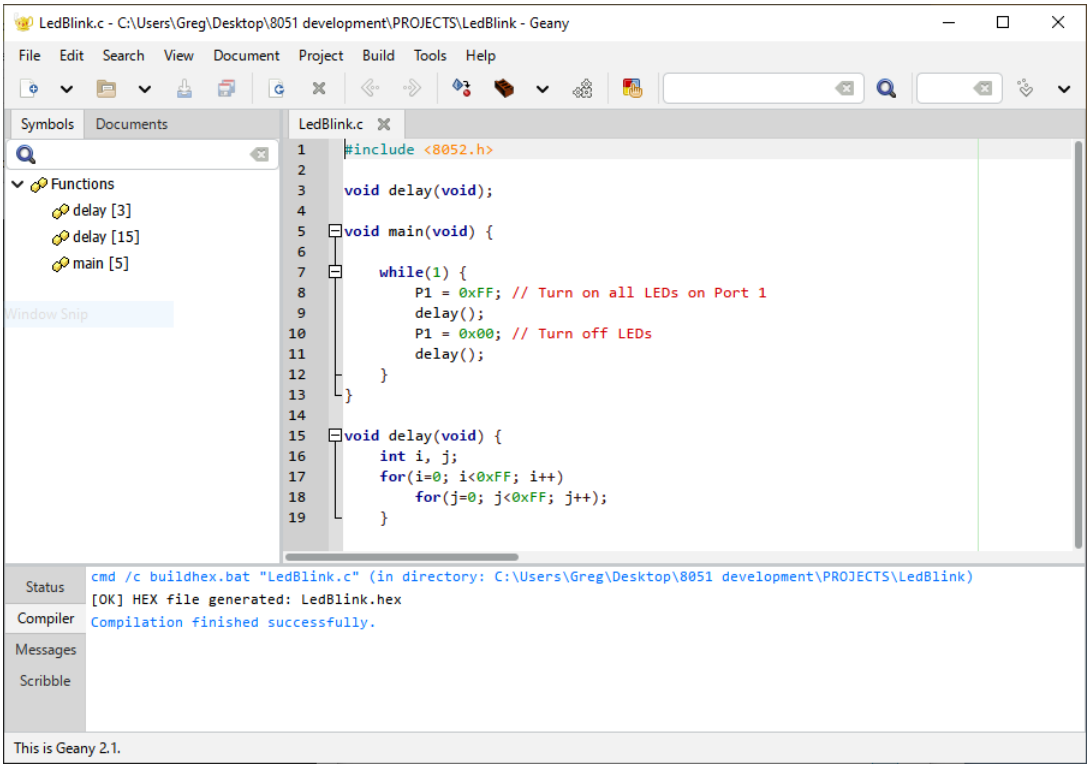
```
Process failed (The system cannot find the file specified)
```

even though the file plainly exists and runs fine when typed at a normal command prompt. Prefixing the command with `cmd /c` opens a shell, lets it interpret the batch file correctly, and then closes – which is the missing piece.

Result

With this in place, the Geany Compiler panel shows the same output as the raw command line:

```
cmd /c buildhex.bat "LedBlink.c" (in directory: ...)
[OK] HEX file generated: LedBlink.hex
Compilation finished successfully.
```



Geany's Compiler panel after a successful Build – the `[OK]` line from `buildhex.bat` confirms the `.hex` file was generated.

F9 (Build) compiles and produces the `.hex`; your Execute shortcut then flashes it, printing the same port/baud banner and waiting for your power-cycle just as it does on the command line.

One More Restart Note

Geany reads the environment (including PATH) once at launch. If you edit your System or User PATH after Geany is already open, close and reopen Geany before testing – otherwise it will keep using the stale PATH it started with, even though a fresh command prompt would show the fix working.

Troubleshooting Summary

- `cc1: unrecognized command-line option '--obj-ext=.rel'` – a non-SDCC `cc1` (usually from a separate MinGW/GCC install) is being found first on PATH. See Section 1.
- `Process failed (The system cannot find the file specified)` **in Geany, but the same command works at a command prompt** – Geany is trying to launch a `.bat` file directly. Prefix the command with `cmd /c`.
- **A PATH fix doesn't seem to take effect** – open a brand new command prompt, and restart Geany if it was already running. Both cache PATH at startup.

Helpful Tips

Interrupt Syntax

SDCC uses double underscores:

```
void func(void) __interrupt(1) {
    ...
}
```

Assembly Support

You may write standalone `.asm` files or embed assembly using:

```
__asm
    ; assembly instructions
__endasm;
```

Cross-Platform

SDCC runs on Windows, Linux, and macOS, though the flashing step above is written for Windows batch files specifically.

Conclusion

You now have a complete, STC89C52RC-specific workflow: compiling with SDCC, producing a flashable `.hex` via `buildhex.bat`, and programming the board over its UART bootloader with `program_hex.bat` – all runnable either from a raw command prompt or from Geany's Build and Execute buttons. From here, the natural next steps are interrupts, timers, and serial communication on real hardware.