

PP, a Small Synthesizable Processor Core

PP is a processor core, very simple, small, and fast,

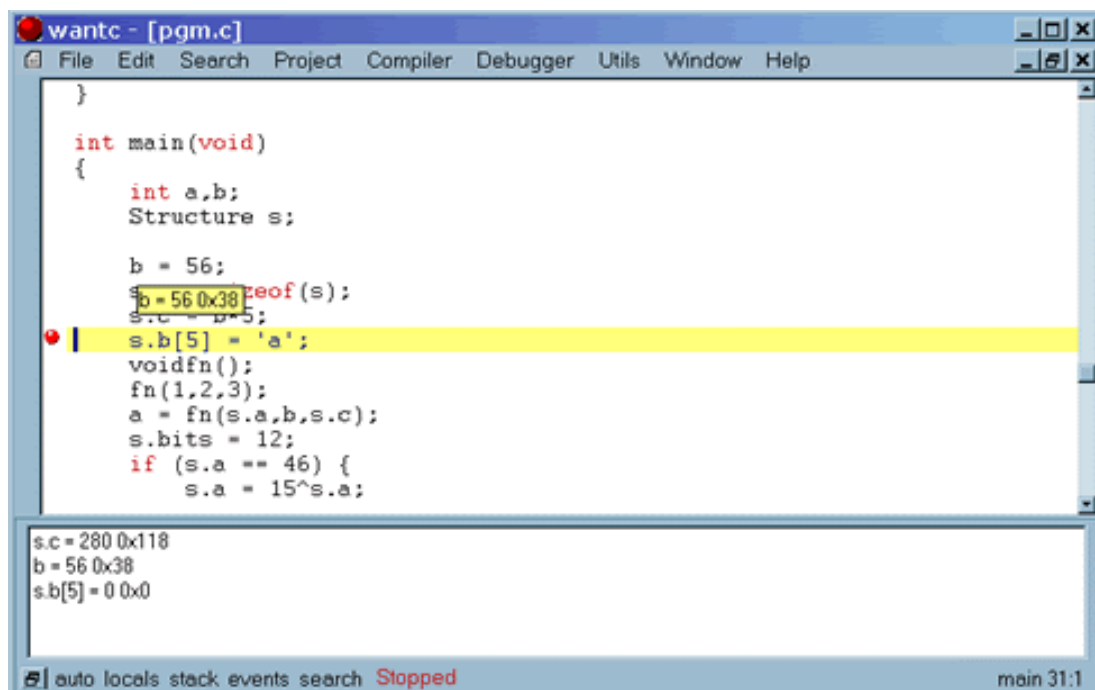
designed for minimum logic and memory footprints, power consumption, and interrupt processing time. Its resources (data path width, stacks depth, memories sizes, ALU functions) and instructions set may be customized to match closely application specific needs.

The PP processor core is specified in synthesizable Verilog,

for easy integration into System-on-Chip of any technology. For easy test, PP core registers are scan-chained through a serial data path, providing full observability and controllability of all PP core and memory-mapped peripheral resources.

The PP processor basic interactive programming tools are free software.

A state of the art integrated development environment in C (LCC) will also be available:



```
wantc - [pgm.c]
File Edit Search Project Compiler Debugger Utils Window Help

}
int main(void)
{
    int a,b;
    Structure s;

    b = 56;
    b = 56 0x38; printf(s);
    s.c = 280 0x118;
    s.b[5] = 'a';
    voidfn();
    fn(1,2,3);
    a = fn(s.a,b,s.c);
    s.bits = 12;
    if (s.a == 46) {
        s.a = 15^s.a;
    }
}

s.c = 280 0x118
b = 56 0x38
s.b[5] = 0 0x0

auto locals stack events search Stopped main 31:1
```

The PP processor is competitive versus other synthesizable processor cores,

thanks to its very small size and high speed (only 1500 cells of the TSMC 0.18um library for 16 bits address and data buses, up to 100 MHz), and mostly thanks to its minimum interrupt overhead (none at interrupt entry, 1 clock cycle when returning from interrupt).

The PP processor is competitive versus wired logic,

when implementing various computations and communication protocols, as for example in the case of intelligent sensors/actuators: wired logic requires different silicon areas for different functions, either on the same chip (more silicon area) or on different versions of the chip (more NRE costs for different fabrication masks), and requires redesign and more NRE costs when the computations or communication protocols evolve, whereas the PP core can simply be reprogrammed, either in flash at no cost, or in ROM at the limited cost of a single mask.

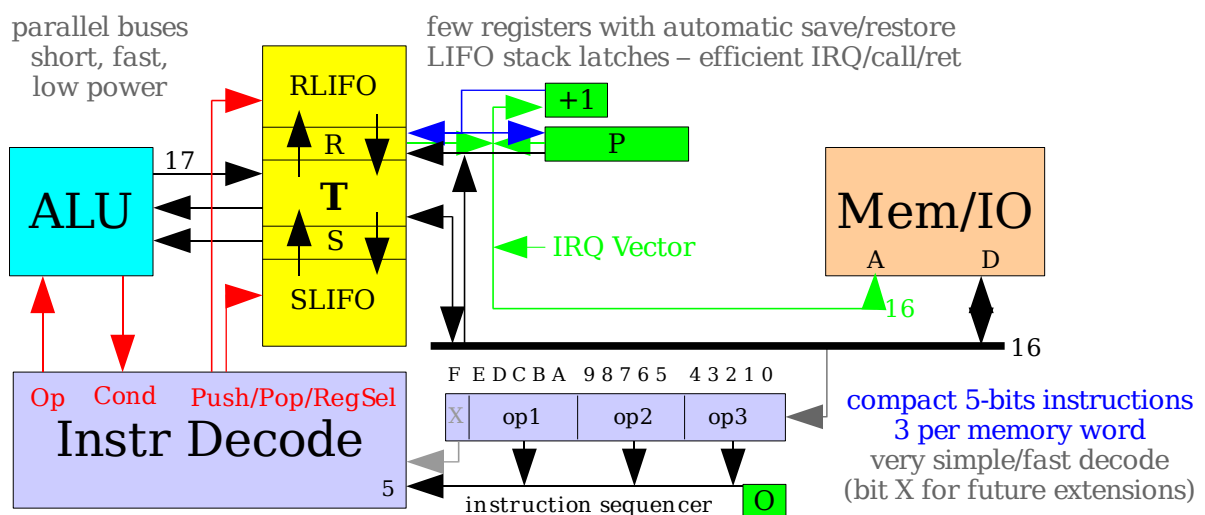
PP core licensing and technical information: contact.christophe.lavarenne@free.fr

PP core architecture

The PP core architecture, for an N bits wide memory data bus, is composed of:

- three main (user-visible) registers R, S, T, each N+1 bits wide; bit N+1 is:
 - . either a redundant sign (same as bit N unless arithmetic overflow)
 - . or a carry for unsigned numbers (no separate carry or other status in the ALU)
 - . or the state of the Global Interrupt Enable (GIE) saved with a return address
- two Last-In-First-Out (LIFO) stacks:
 - . an address stack (typically 8 cells deep) prolonged by the register R
 - . a data stack (typically 8 cells deep) prolonged by registers S and T
 - . they automatically save/restore the 3 main registers when written/read:
this saves instructions, and mostly interrupt processing time
- an Arithmetic and Logic Unit (ALU) directly connected to the 3 main registers:
 - . concurrently computes all its outputs (AND, OR, XOR, SUM, etc.)
 - . propagates its sum carry concurrently with the instruction decoding
 - . the instruction selects for each main register one ALU output, or another source, to be sampled at end of clock cycle
- an instruction decoder for a set of 32 instructions coded over 5 bits
 - . every memory word (N bits) stores N/5 instructions, or one N-bits literal
 - . the instruction register I stores N/5 instructions read in one cycle from memory, either by the program pointer P, or by the register R (branch or return)
 - . every instruction in I, indexed by register O, executes in a single cycle
 - . T or R may be loaded from memory addressed by P post-incremented
 - . T may be loaded from or stored to memory addressed by R, post-incremented or popped

The following diagram illustrates the PP architecture for N=16:



NOP	NOT	ASL	ASR	RDP	DRP	SWP	XCH
OVR	DUP	S0	S1	ADD	XOR	IOR	AND
STP	STI	LDP	LDI	RFR	TOR	MUL	DIV
LNK	ADR	LDN	LIT	BR	BNC	BZ	BNZ

Instruction set: 1 clock cycle per instruction
18 general opcodes to code any algorithm,
14 dedicated opcodes for code optimizations
 (such as multiply-step, divide-step, etc.)