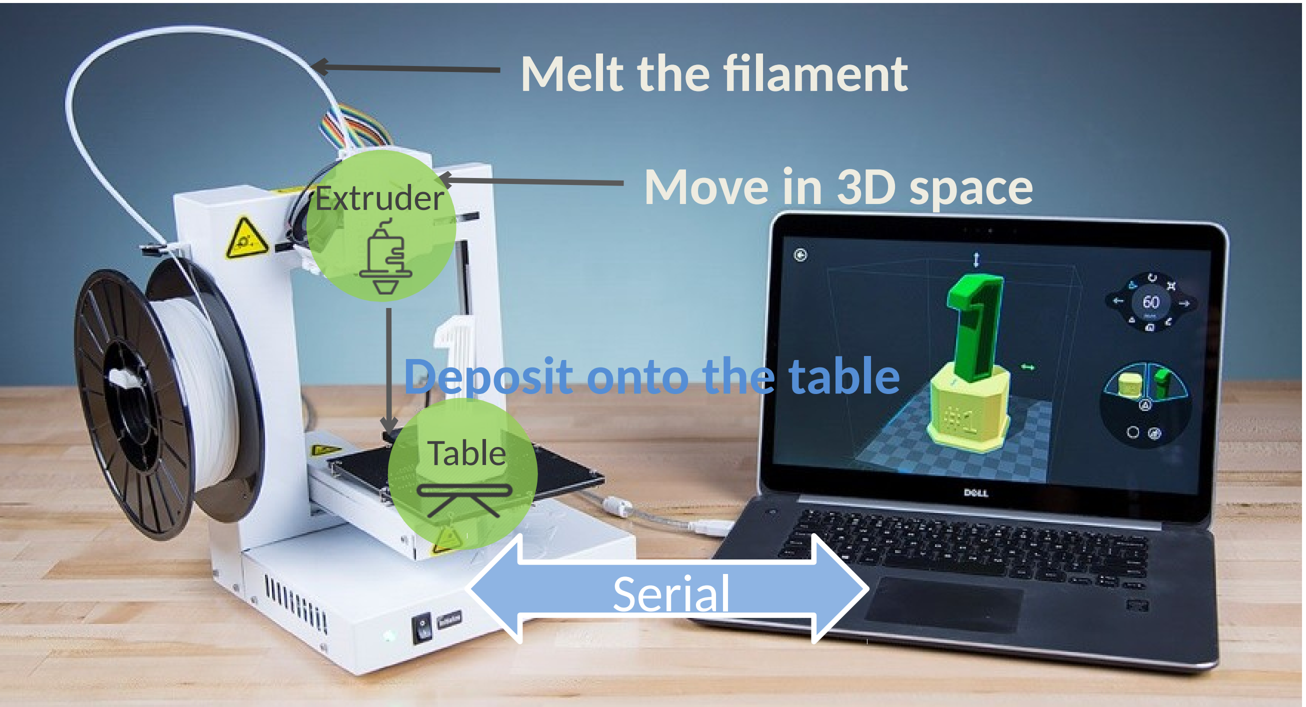
Building Real-Time Embedded Applications on QduinoMC

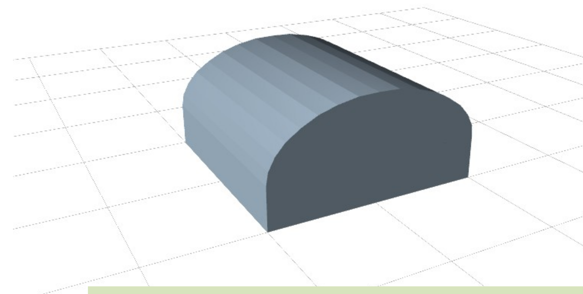
A Web-connected 3D Printer Case Study

Zhuoqun (Tom) Cheng, Richard West and Ying Ye

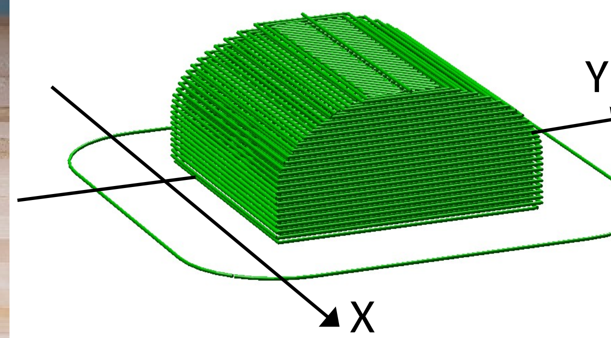
3D Printing HOW-TO



CAD Model

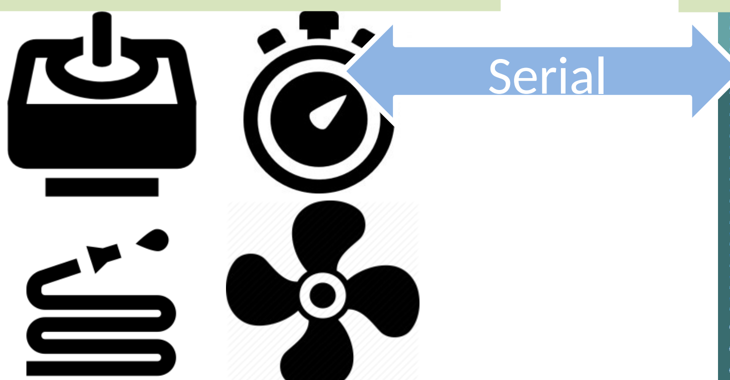


Slicing Program



Printed Object

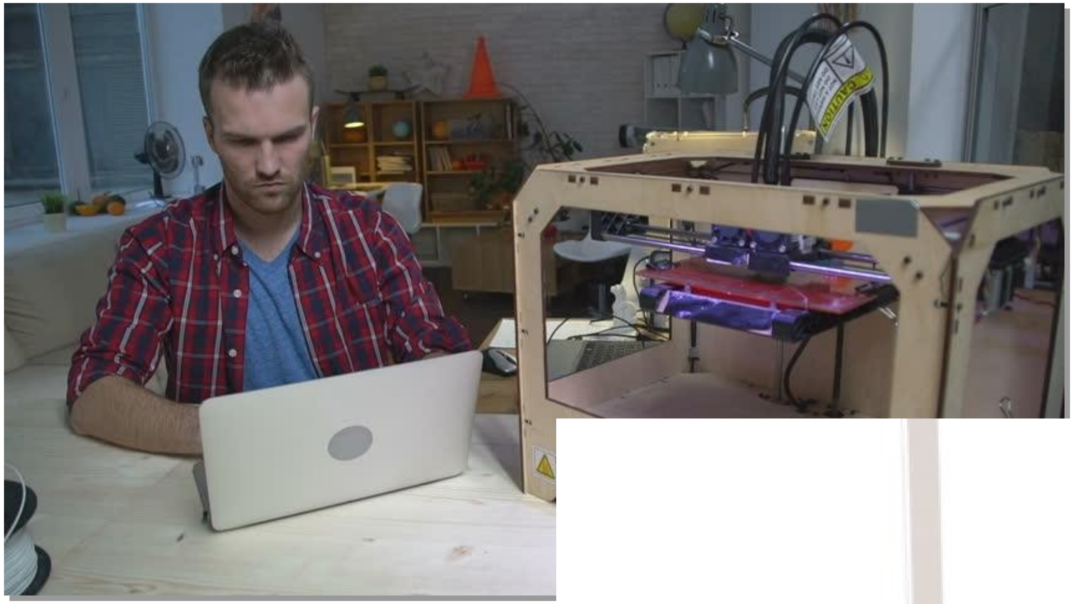
Firmware



G-code

```
;Layer count: 336
;LAYER:0
M107
G0 F9000 X91.800 Y93.520 Z0.300
;TYPE:SKIRT
G1 F1200 X92.617 Y92.870 E0.01964
G1 X93.518 Y92.412 E0.03865
G1 X94.458 Y92.141 E0.05705
G1 X95.218 Y92.072 E0.07141
G1 X95.998 Y92.064 E0.08608
G1 X96.894 Y92.071 E0.10294
G1 X98.900 Y92.070 E0.14067
G1 X100.514 Y92.071 E0.17103
G1 X101.565 Y92.065 E0.19080
G1 X102.277 Y92.081 E0.20420
G1 X102.421 Y92.091 E0.20691
G1 X103.374 Y92.066 E0.22484
```

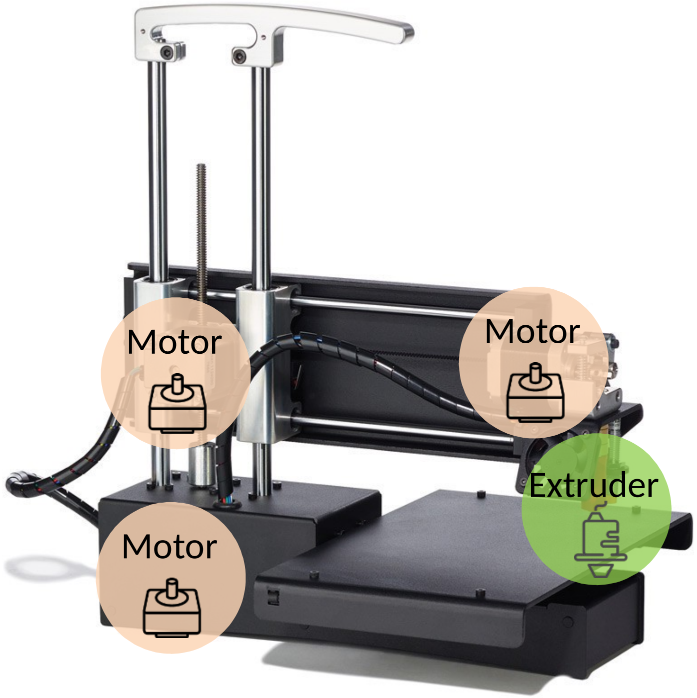
Web-connected 3D Printer

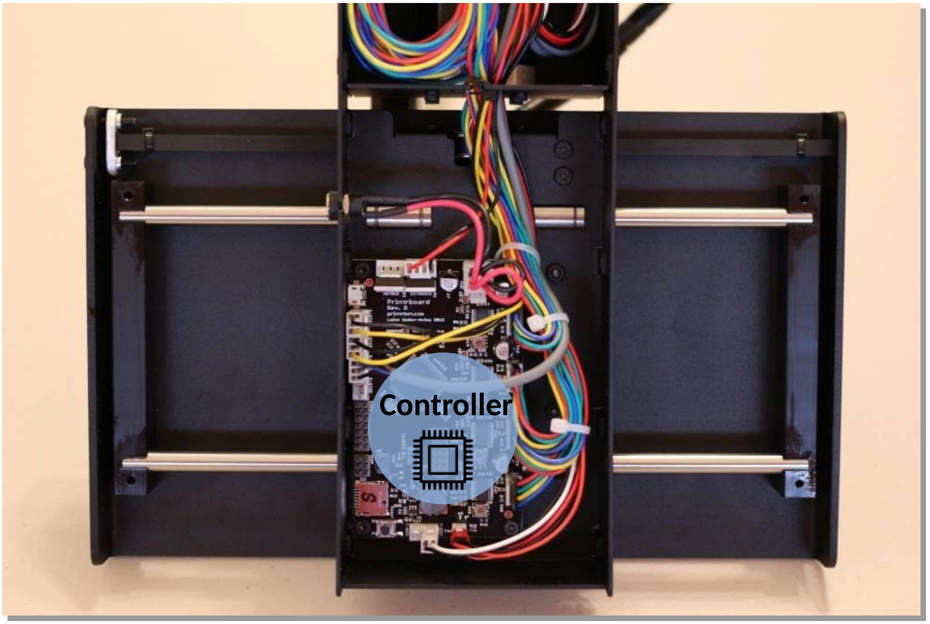


Web-connected 3D Printer

- Remote Job Submission
- Local Slicing
- Correctness Verification

Printrbot Simple Metal





Web Server

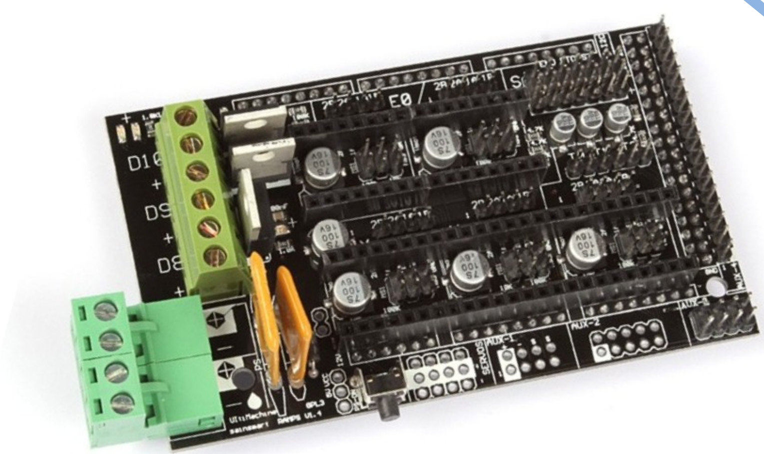
Microprocessor	Atmel AVR, 8 bit, 20 MHz
SRAM	8 KB
I/O	UART, SPI, I2C, PWM, GPIO

Custom Controller

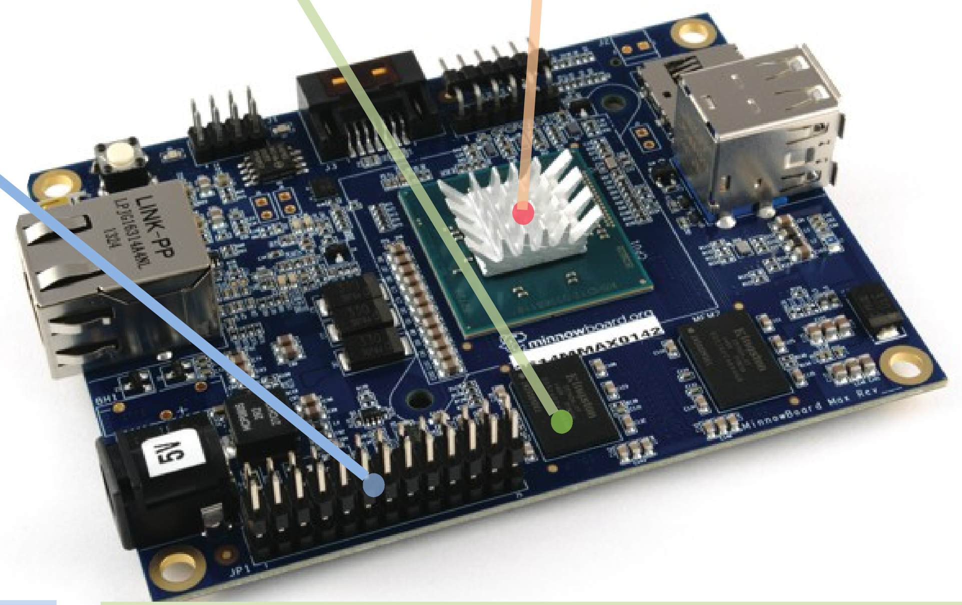
86 GPIOs, I2C, SPI, UART, PWM

2 GB memory

64 bit dual-core Atom E3825 1.33 GHz



RAMPS shield: I/O extension board



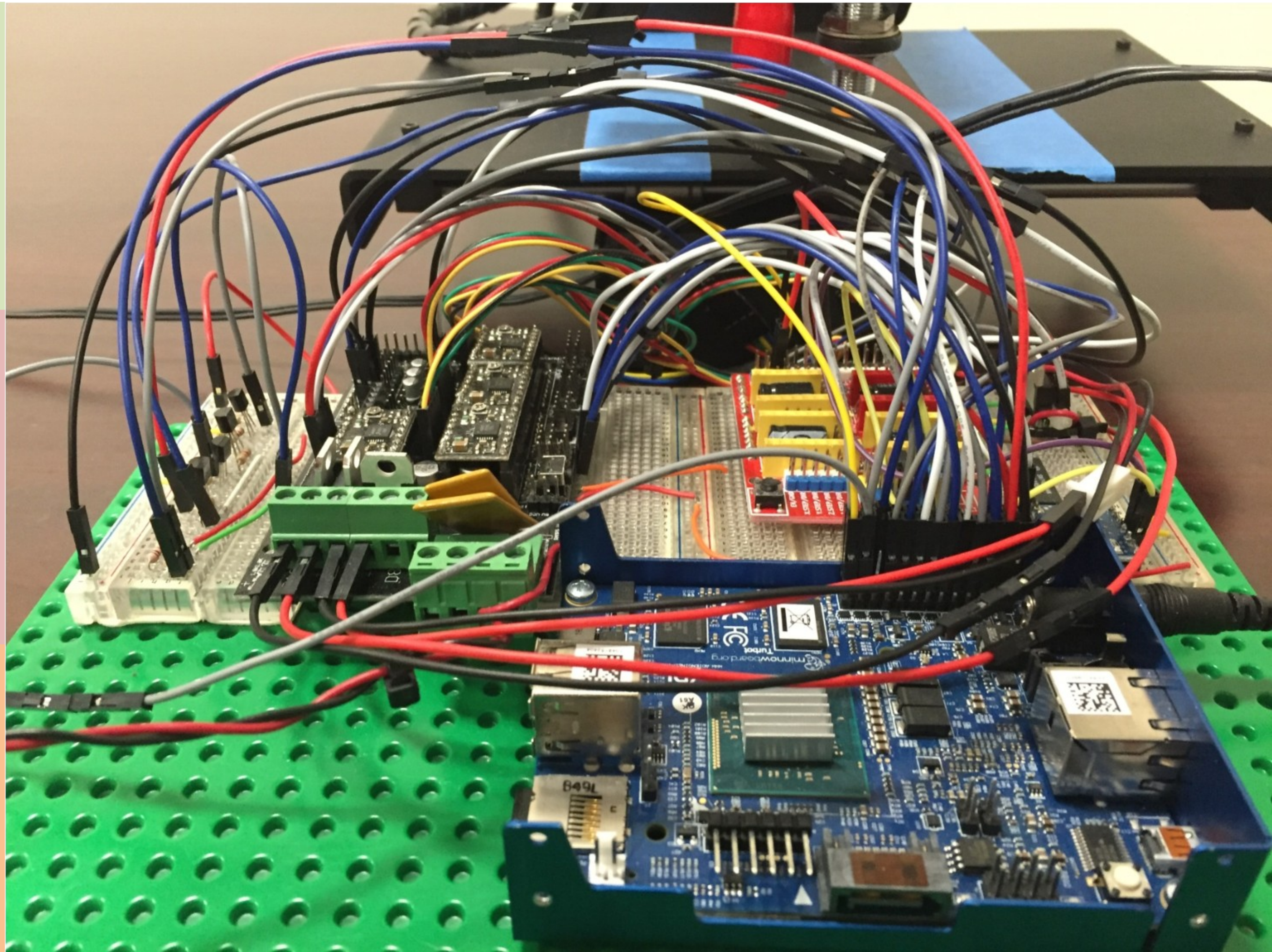
Intel MinnowBoard MAX

Custom Controller

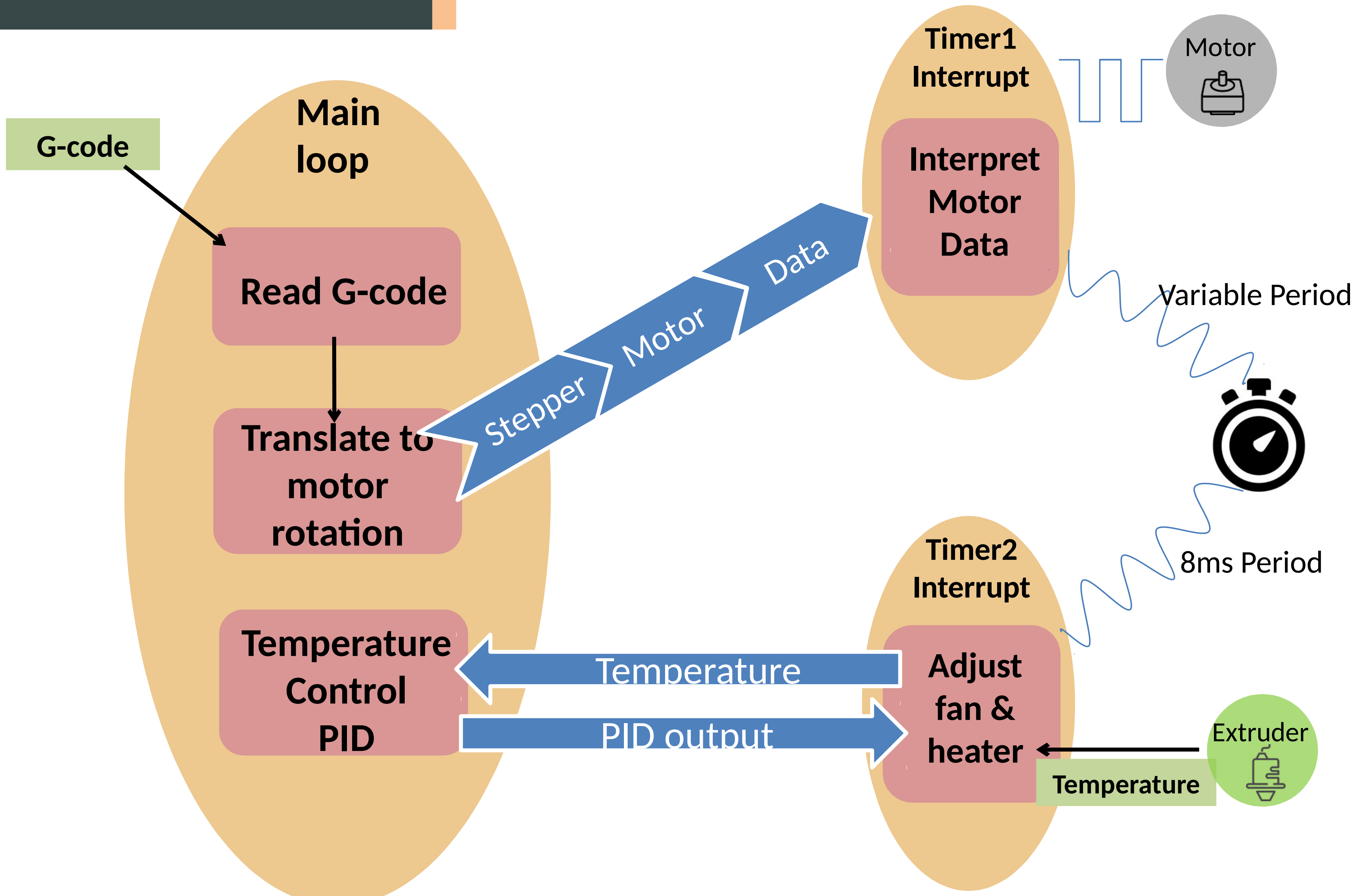
RAMPS shield

Companion Analog Circuits

MinnowMAX



Marlin Firmware

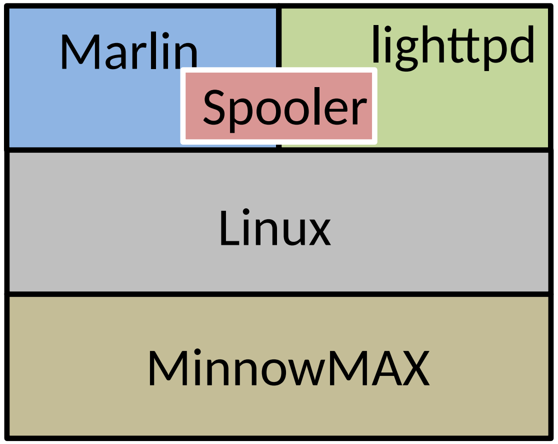


Marlin on Linux

Original Marlin
Main loop + interrupts handlers
Timer interrupts
AVR I/O instructions
All computations in the main loop

Jitter of the extruder, when submitting relatively large files

Is this bad? Why?



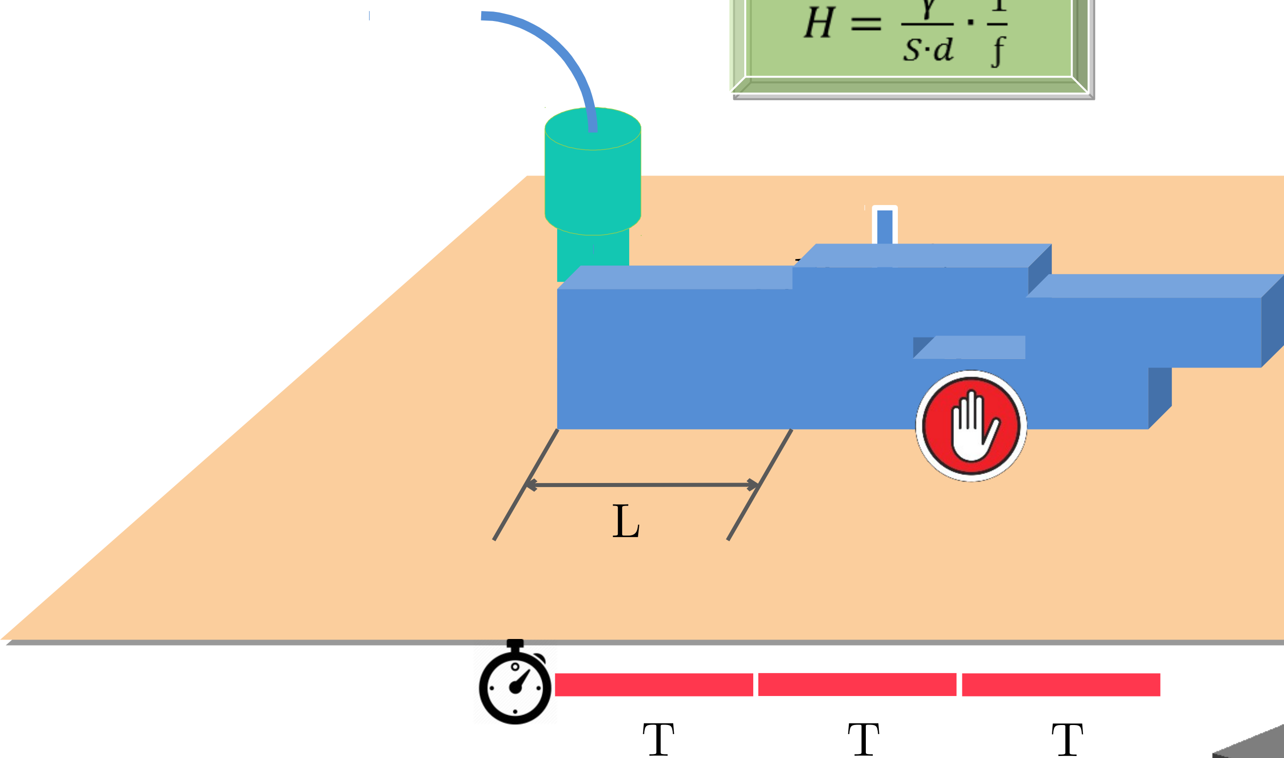
The Timing Problem

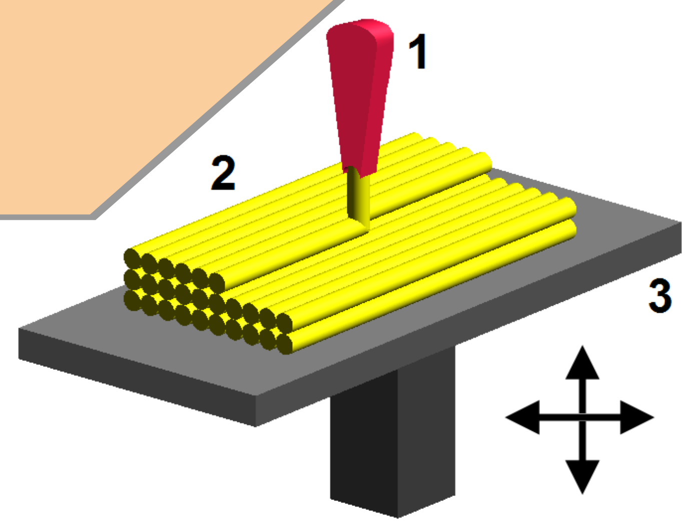
$$Volume = \gamma \cdot T = L \cdot H \cdot d$$
$$= f \cdot T \cdot S \cdot H \cdot d$$

f -- pulse frequency

S – linear displacement per pulse

$$\boxed{H = \frac{\gamma}{S \cdot d} \cdot \frac{1}{f}}$$





10 kHz Pulse Train

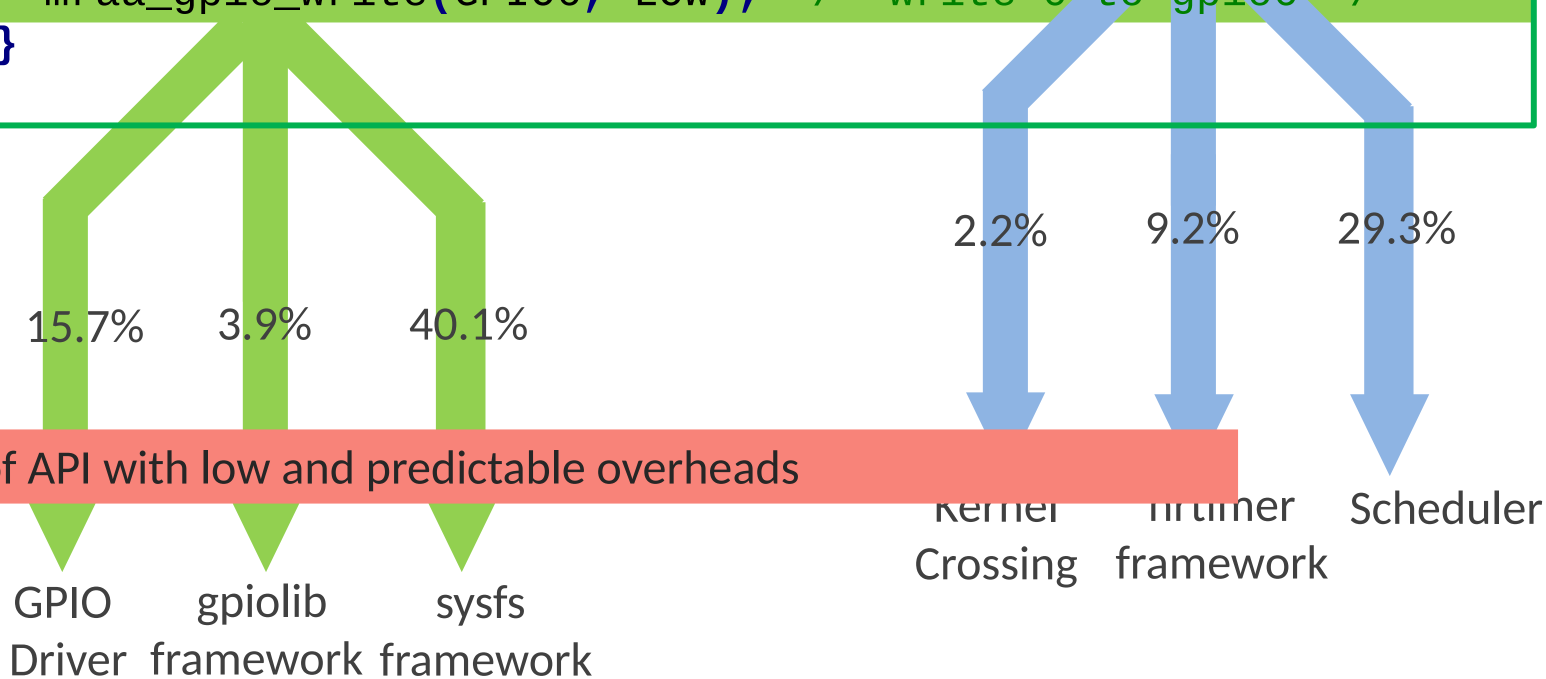
```
struct timespec period = {.tv_sec = 0, .tv_nsec = 100000};
while (1) {
  nanosleep(&period, NULL);     /* sleep for 100 us */
  mraa_gpio_write(GPIO6, HIGH); /* write 1 to gpio6 */
  mraa_gpio_write(GPIO6, LOW);  /* write 0 to gpio6 */
}
```

	Frequency	Period
Theoretical	10 kHz	100000 ns
Measured	7.91 kHz	100000 ns + 26422 ns
Original PrintrBoard	9.96 kHz	100000 ns + 401 ns

Unstable

10 kHz Pulse Train

```
struct timespec period = {.tv_sec = 0, .tv_nsec = 100000};
while (1) {
  nanosleep(&period, NULL);      /* sleep for 100 us */
  mraa_gpio_write(GPIO6, HIGH); /* write 1 to gpio6 */
  mraa_gpio_write(GPIO6, LOW);  /* write 0 to gpio6 */
}
```



Lack of API with low and predictable overheads

10 kHz Pulse Train

```c
struct timespec period = {.tv_sec = 0, .tv_nsec = 100000};
while (1) {
  nanosleep(&period, NULL);      /* sleep for 100 us */
  mraa_gpio_write(GPIO6, HIGH); /* write 1 to gpio6 */
  mraa_gpio_write(GPIO6, LOW);  /* write 0 to gpio6 */
}
```

Real-time environment setup
PREEMPT_RT patch
Setting scheduling priority
Locking pages into memory
Further optimization
Shield a core from the scheduler: isolcpu, cset…
Disable timer interrupt on a core: CONFIG_NO_HZ_FULL
….

Lack of a simple and uniform programming interface

QduinoMC

Goals	Design
Easy to use	
Easy to port existing Arduino program	
Take advantage of the multi-core	
Allow QoS specification	
Low I/O access overhead	

loop (loopID, budget, period, [coreID])

noInterrupts (device, coreID)
noTimer (coreID)

interruptsVcpu (device, budget, period, [coreID])

digitalWrite () / digitalRead ()

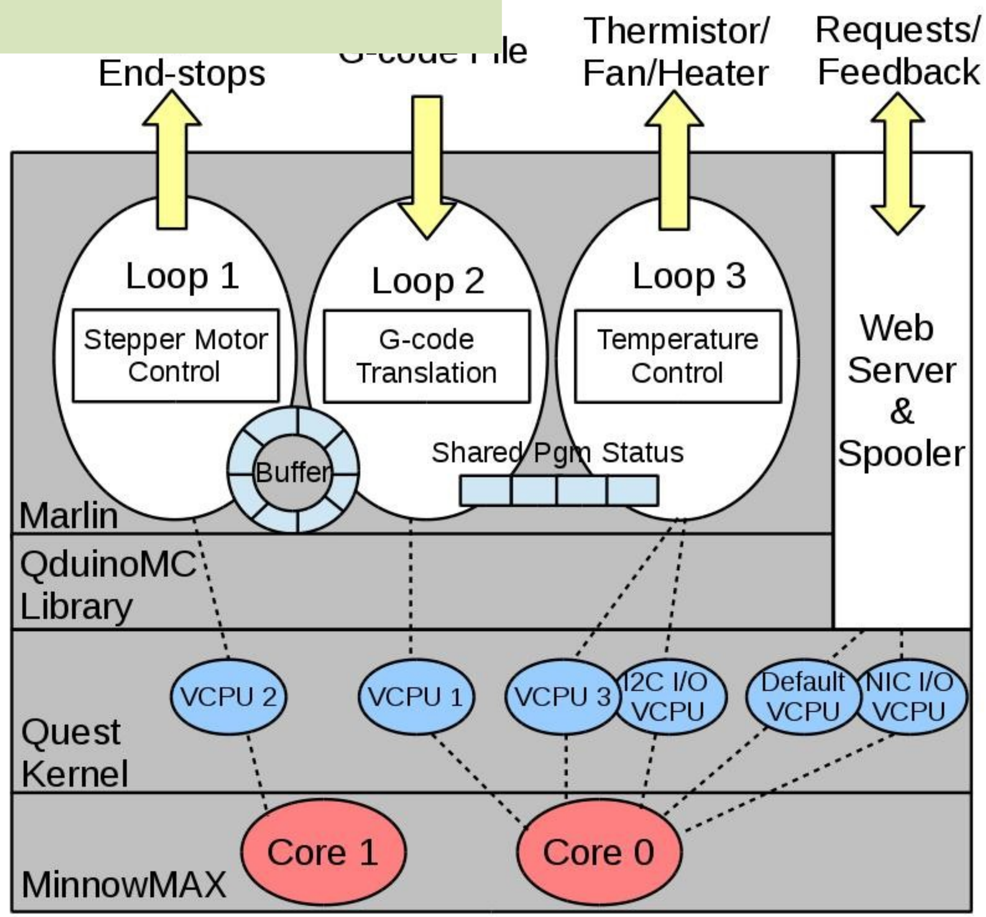
Marlin on QduinoMC

loop (1, 10, 100, 1), loop (2, 30, 100, 0), loop (3, 1, 80, 0)

interruptsVCPU (I2C, 10ms, 100ms), interruptsVCPU (NIC, 10ms, 100ms)

noTimer (1), noInterrupts (ALL, 1)

Web server / Spooler -- default



10 kHz Pulse Train Again

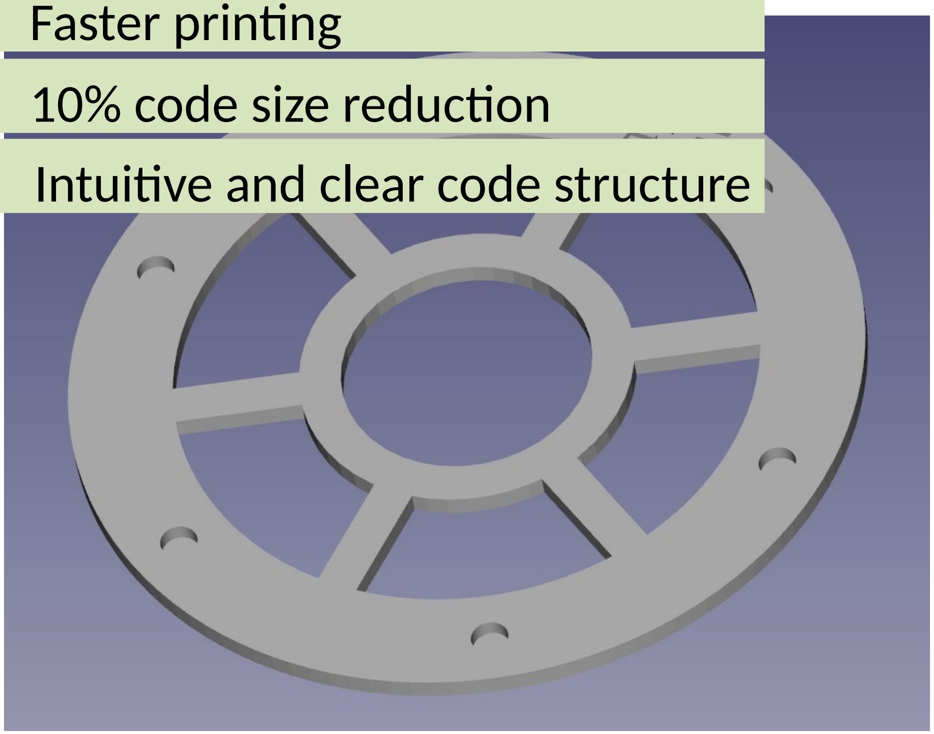
```
void setup ( ) {
 pinMode(GPIO6, OUTPUT);
 noInterrupts(ALL, 1); noTimer(1);
}

void loop (1, 100, 100, 1) {
 delayBusyNanoseconds(100000);
 digitalWrite(GPIO6, 1); digitalWrite(GPIO6, 0);
}
```

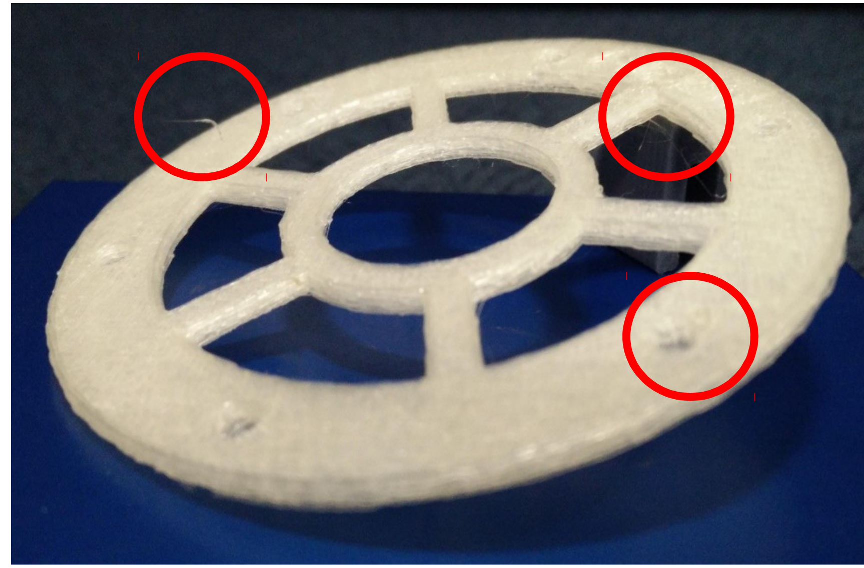
	Frequency	Period
Theoretical	10 kHz	100000 ns
Measured QduinoMC	9.569 kHz	100000 ns + 4504 ns
Measured Linux	7.91 kHz	100000 ns + 26422 ns
Original PrintrBoard	9.96 kHz	100000 ns + 401 ns

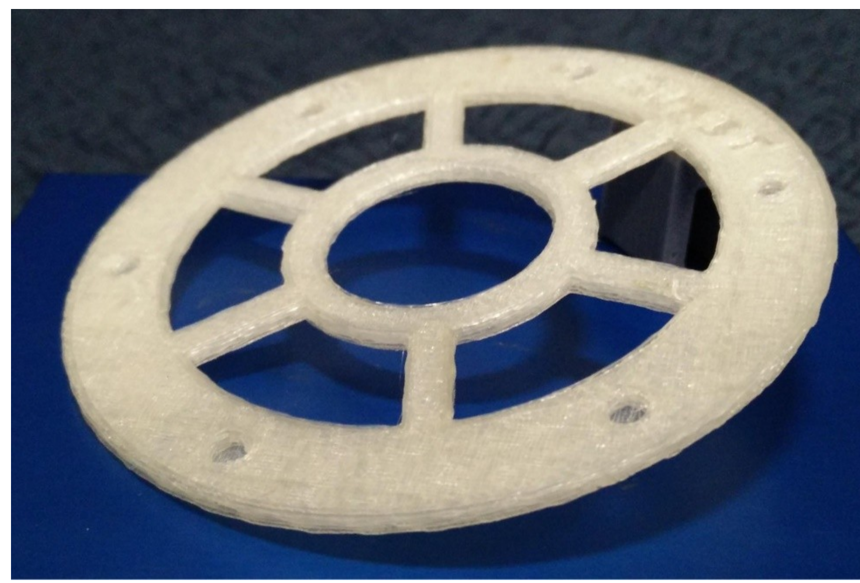
Stable

Test Object

- Higher quality
- Faster printing
- 10% code size reduction
- Intuitive and clear code structure







Conclusion & Future Work

Designed and built a platform to ease the development of IoT applications with critical timing requirements

Built a web-connected 3D printer as a case study
analyzed 3D printers' real-time properties

Future work will extend web connectivity to support local slicing and print verification

Thank you!

Questions?