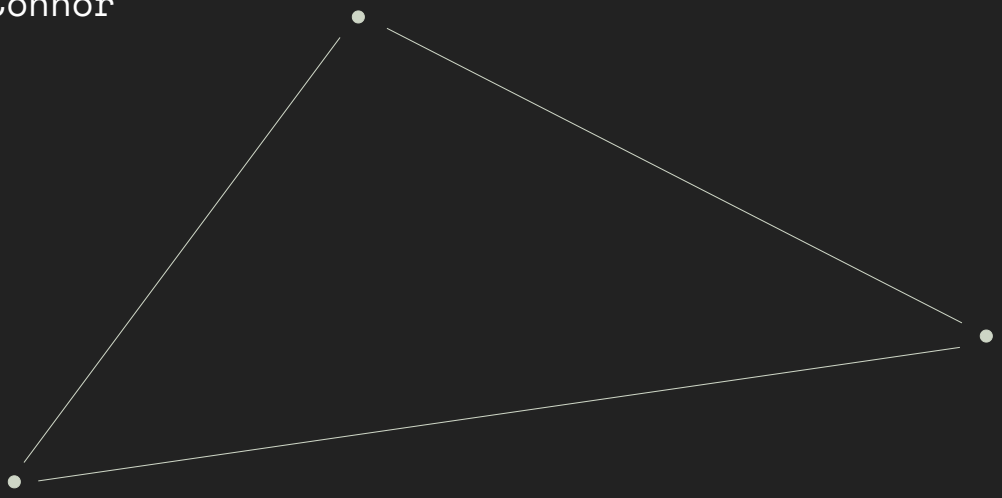
CLOCKWORK

Clockwork And Its Tools - Open Source Software to make things more easily

Mike O'Connor



LATPROC TEAM

Mike O'Connor

- the person with the know-how
- the person with the need

Martin Leadbeater

- @mleadbeater
- the helpful designer/programmer

LATPROC PROJECTS

clockwork - language and main toolset

https://github.com/latproc/clockwork

scope - an event sampler and terminal oscilloscope

https://github.com/latproc/scope

humid - Human Interface Daemon

https://github.com/latproc/humid

clockwork_esp32 - ESP32/FreeRTOS runtime

https://github.com/latproc/clockwork_esp32

LATPROC

Language And Tools for Process Control

- First published 2012
- Linux Framework
- FreeRTOS Framework (tested on ESP32)
- Open Source under BSD and GPL

ORIGINAL DESIGN CAPABILITIES

Industrial IO (real world connections)

- BeckHoff EtherCAT(tm) based control

Humid (Human Machine Interface)

Protocols

- Modbus master/slave
- - Modbus TCP/IP and RTU (Serial)
- Raw TCP/IP and serial UART (device_connector)
- Plugins
- - libcurl
- - exec system
- MQTT - IoT protocol

WHAT IT WAS DEVELOPED FOR

WHAT IT WAS DEVELOPED FOR

CLOCKWORK

The latproc language

Objects are called Machine's

Describe Machines by their states

States are selected by evaluating rules

All Machines run continuously in parallel

Machines monitor selves and each other

LANGUAGE FEATURES

Rule based

Automatic state selection

No Loop statements

Highly reusable code

DEFINING STATE MACHINES

```
Light MACHINE {
        on STATE;
        off INITIAL;
}
light Light;
```

Simple machine

No automation

Controlled externally

'MACHINE' is a bit like a 'Class'

Instantiate a MACHINE to use it in a program

USING TRANSITIONS

```
ToggleSwitch MACHINE {
        on STATE;
        off INITIAL;

        TRANSITION on TO off USING next;
        TRANSITION off TO on USING next;
}
```

Basic automation via transitions

Flips state when 'next' message is received

SITUATION NORMAL

MACHINEs can have STATEs

TRANSITIONS can cause STATEs to change

Nothing special so far...

STATE MONITORING

Our name for this programming style

State Machines are a common technique

State Machines focus on events and transitions

State Monitoring is about states and rules

USING RULES

```
Blinker MACHINE {
        on WHEN SELF IS off;
        off DEFAULT;
}
```

Automatic state selection

Rules (WHEN statements) are continuously checked

Evaluates rules in sequence

Stops at the first rule that is true

CONTROLLING THINGS

```
LightSwitch MACHINE switch, light {
        on WHEN switch IS on;
        off DEFAULT;

        ENTER on  { SET light TO on; }
        ENTER off { SET light TO off; }
}
```

Controls 'light' based on the state of 'switch'

Rules (WHEN statements) are continuously checked

ENTER functions are processed when a state changes

The SET statement tells a machine to change state

NO LOOPS OR IFS?

```
Counter MACHINE {
        OPTION count 0;

        counting_up WHEN count < 10;
        idle DEFAULT;

        ENTER counting_up { count := count + 1; }
}
```

This machine enters 'counting_up' whenever count < 10

When it enters 'counting_up' the machine adds one

NO IF STATEMENTS?

```
Counter MACHINE {
        OPTION count 0;
        up WHEN count < 10;
        idle DEFAULT;

        ENTER up {
                count := count + 1;
                IF count < 10 { SET SELF TO idle; }
        }
}
```

An ugly fix to the error in the rules

COUNTING UP

```
Counter MACHINE {
        OPTION count 0;
        up WHEN SELF IS idle AND count < 10;
        idle DEFAULT;

        ENTER up { count := count + 1; }
}
```

A nicer fix to the previous bug

Generally avoid using IF, use rules instead

REUSING COMPONENTS

```
LightSwitch MACHINE switch, light {
        on WHEN switch IS on;
        off DEFAULT;

        ENTER on  { SET light TO on; }
        ENTER off { SET light TO off; }
}
```

- Passing machines as parameters links them
- switch and light are available to LightSwitch
- LightSwitch can monitor the state or properties of linked machines

CONTROLLING MULTIPLE THINGS AT ONCE

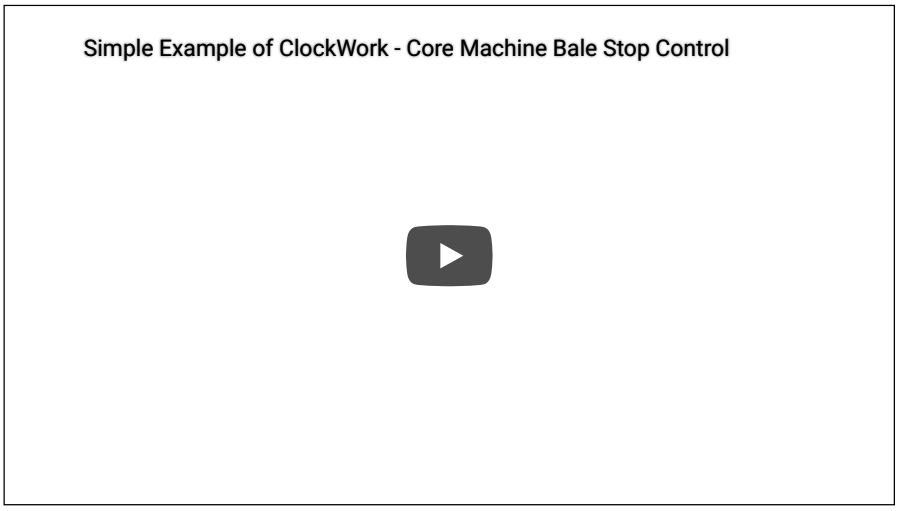
```
LightController MACHINE lights {
        COMMAND on  { SEND turnOn TO lights; }
        COMMAND off { SEND turnOff TO lights; }
}

front_porch Light;
garden Light;
yard_lights LIST front_porch, garden;

controller LightController yard_lights;
```

A LIST 'machine' can hold several other machines

When a message is sent to a LIST it is redirected to the members

CLOCKWORK CODE REAL-WORLD EXAMPLE

Wool bales being cored



A COMPARISON OF PROGRAMMING STYLES

Programming blinking LEDs in C and Clockwork

Blinking LEDs - the hello world of hardware

Programming hardware starts easy but gets hard

BLINK A LED

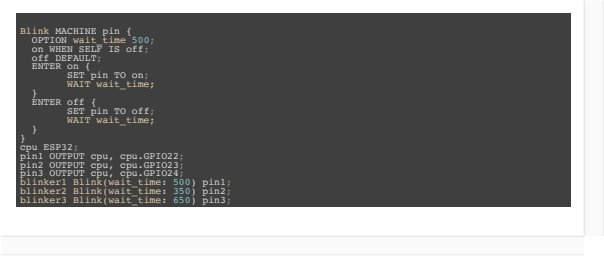
```
int pin = 6;
byte level = HIGH;
void setup() {
  pinMode(pin, OUTPUT);
  digitalWrite(pin, level);
}

void loop() {
  level = (level == LOW) ? HIGH : LOW;
  digitalWrite(pin, level);
  delay(500);
}
```

```
Blink MACHINE pin {
  on WHEN SELF IS off;
  off DEFAULT;
  ENTER on {
        SET pin TO on;
        WAIT 500;
  }
  ENTER off {
        SET pin TO off;
        WAIT 500;
  }
}
cpu ESP32;
led OUTPUT cpu, cpu.GPIO22;
blinker Blink led;
```

A barebones example of code in C for Arduino and in Clockwork for ESP32

ADD MORE LEDS

```
typedef struct {
  int pin;
  unsigned long last_time;
  byte level;
  int wait_time;
} Led;

Led pins[] = {
  {. pin = 6, .last_time = 0, .level = LOW, .wait_time = 500},
  {. pin = 7, .last_time = 0, .level = LOW, .wait_time = 350},
  {. pin = 8, .last_time = 0, .level = LOW, .wait_time = 650}
};

void setup() {
  int i;
  unsigned long now = millis();
  for (i=0; i<3; ++i) {
    pinMode(pins[i].pin, OUTPUT);
    digitalWrite(pins[i].pin, pins[i].level);
    pins[i].last_time = now;
  }
}

void loop() {
```



```
Blink MACHINE pin {
  OPTION wait_time 500;
  on WHEN SELF IS off;
  off DEFAULT;
  ENTER on {
        SET pin TO on;
        WAIT wait_time;
  }
  ENTER off {
        SET pin TO off;
        WAIT wait_time;
  }
}
cpu ESP32;
pin1 OUTPUT cpu, cpu.GPIO22;
pin2 OUTPUT cpu, cpu.GPIO23;
pin3 OUTPUT cpu, cpu.GPIO24;
blinker1 Blink(wait_time: 500) pin1;
blinker2 Blink(wait_time: 350) pin2;
blinker3 Blink(wait_time: 650) pin3;
```

Not a big change in the requirements

Quite a lot of code changes in C

A NICER VERSION

We rarely use the WAIT statement

```
Blink MACHINE pin {
  OPTION wait_time 500;
  on WHEN SELF IS off AND TIMER >= wait_time;
  on WHEN SELF IS on AND TIMER <= wait_time;
  off DEFAULT;
  ENTER on  { SET pin TO on;  }
  ENTER off { SET pin TO off; }
}
```

Rules can be split so they can be expressed naturally

"If you've been off for 500ms or more, turn on"

"If you're on stay that way for 500ms"

We like to keep the ENTER functions small

TOOLS

iod - main EtherCAT control daemon

cw - local interpreter daemon

iosh - terminal shell

sampler - monitoring and logging

persistd - monitor and retain persistend state

modbusd - bridge to modbus masters (panels)

device connector - bridge to TCP/IP or serial devices

TOOLS

iosh

```
> DESCRIBE O24V_GrabControlFan ;
--------------
O24V_GrabControlFan: off  Class: POINT
 instantiated at: /opt/latproc/code/config/config.lpc line:94
  parameter 1 module (EL2828_01), state: OP
  parameter 2 1 ()

 io: readtime: 185932 [Channel 2 Output, 1 0:1.1]=0 (0)
Exports:
  O24V_GrabControlFan
  published (1)
Listening to:
  EL2828_01[123]:   OP

Dependant machines:
  L_Inputs[399]:   nonempty
  M_ControlFan[580]:   off

properties:
  NAME:O24V_GrabControlFan, STATE:off, export:0, tab:Outputs, type:Output, wire:Y52

Timer: 20713486

>
```


TOOLS

sampler

```
443391 M_CO2 idle
443891 M_CO2 update
443892 XA_CO2 CO2_Level 496
443892 XA_CO2 message 496
443892 M_CO2 idle
```

USER INTERFACE TOOLS

Web UI - basic web application for monitoring and control

humid - GUI development tool

- Buttons (toggle, momentary)
- Text and Number Entry Fields
- Page switching
- Element visablity control
- Time series graphing

web 3D visualisation - render a model in 3D

scope - character based graphing for sampler

EMBEDDING CLOCKWORK

Automatically translate user code to C

Provides a runtime framework

Limited set of features currently built

Intending to support Arduino (AVR ATmega) devices

Intending to support other low power devices

Working on better integration with our Linux based toolset

EMBEDDED FEATURE SET

Basic runtime working

No string processing

No floating point calculations

No iosh, debug is via mosquitto_pub/sub

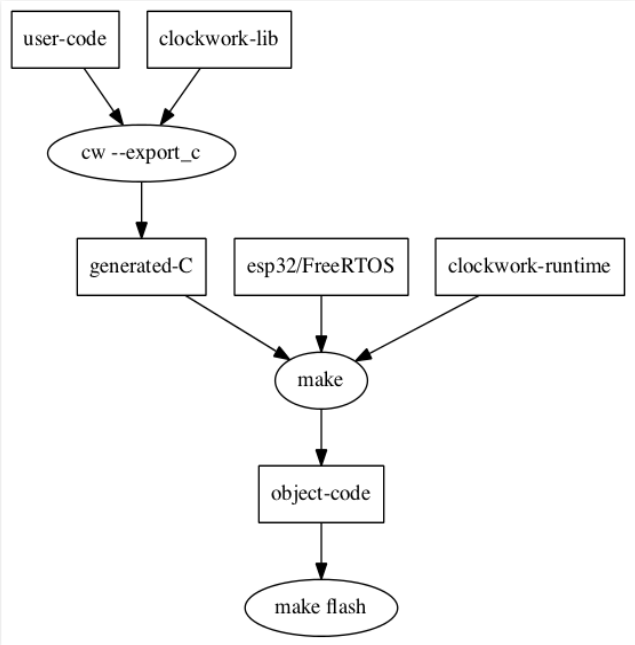
Humid not yet supported

BUILD PROCESS

Command based

Work from clockwork source directory

```
cd my-project
build_esp32 [-f]
```



OPEN SOURCE COMPONENTS

Linux, GNU compiler suite

cJSON, libmodbus, ZeroMQ

EtherLAB, libXML, three.js

nanogui, glfw

etc.,

BUGS AND ISSUES

Not enough documentation

Numerous small issues

Inter-clockwork channels needs more work

- Shadowing machines
- Startup issues when components restart

Some language features need clarification

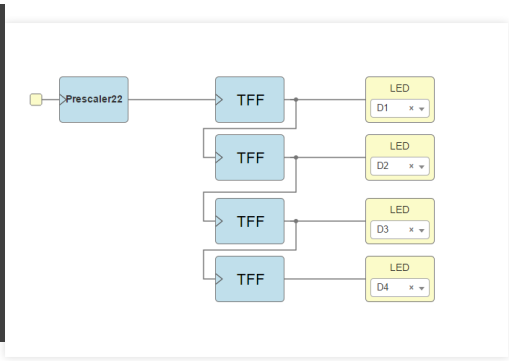
APPLICATION TO FPGAS

Can do some high-level digital simulation

One day we would like to program FPGAs

An example: 4-bit counter:

```
Pulser MACHINE {
        OPTION delay 1000;
        on WHEN TIMER > delay;
        off DEFAULT;
}
TFF MACHINE in {
        on STATE; off INITIAL;
        RECEIVE in.off_enter { SEND next TO SELF; }
        TRANSITION on TO off ON next;
        TRANSITION off TO on ON next;
}
pulse Pulse;
one TFF pulse;
two TFF one;
three TFF two;
four TFF three;
```



MONITORING WITH SAMPLER

Using the 4-bit counter example

sampling to watch the flip flop changes

```
11150   one     off
11152   two     off
11156   three   off
11160   four    off
11174   monitor.VALUE   value   0
12159   one     on
12188   monitor.VALUE   value   1
13166   one     off
13169   two     on
13192   monitor.VALUE   value   2
14172   one     on
14198   monitor.VALUE   value   3
15178   one     off
15181   two     off
15185   three   on
15207   monitor.VALUE   value   4
16190   one     on
```

BIT #				Result
four	**three**	**two**	**one**	**value**
off	off	off	off	0
			on	1
		on	off	2
		on	on	3
	on	off	off	4

CONVERT 4-BITS TO A NUMBER

Machines can be placed in a List

A List can interpret the on-off state of its members to create a BITSET

A BITSET can be interpreted as a number

BITSETS can be saved and loaded and used to initialise members of list to a known state.

```
res LIST four, three, two, one;
Monitor MACHINE list {
  OPTION VALUE 0;
  update WHEN VALUE != BITSET FROM list;
  idle DEFAULT;
  ENTER update { WAIT 20; VALUE := BITSET FROM list; }
}
monitor Monitor res;
```


FUTURE

We are continuing to develop and refine the language

Tools are developed as we need them

Will use the compiler technology on Linux

Lots of other ideas on the drawing board

We would like to help others use the tool set

QUESTIONS OR COMMENTS

clockwork - language and main toolset

https://github.com/latproc/clockwork

scope - an event sampler and terminal oscilloscope

https://github.com/latproc/scope

humid - Human Interface Daemon

https://github.com/latproc/humid

clockwork_esp32 - ESP32/FreeRTOS runtime

https://github.com/latproc/clockwork_esp32

QUESTIONS OR COMMENTS

Latproc - Language and Tools for Process Control

- https://github.com/latproc