Pacemaker CIS541

ARIEL EIZENBERG

arieleiz@seas.upenn.edu

Project Highlights - Pacemaker

- A DDD-R pacemaker using an **atrial-based lower rate timing (A-A interval)** modelled in UPPAAL, and a heart model used for correctness testing.
- The pacemaker includes several extra features not required by the project specification:
  - PVAB timing cycle,
  - PVC support (required by A-A interval),
  - Dynamic AVI extension,
  - Alarm buzzer

Project Highlights – Code Generation

- Developed a versatile UPPAAL to MBED C++ automatic code generator
  - Direct support for binding UPPAAL channels to hardware,
  - Accepts UPPAAL XML files directly,
  - Supports most of the UPPAAL language,
  - Verified with several models (incl. from class),
  - Used to generate code for the pacemaker and heart.
- The auto-generated code was used with no changes, as-is, for the final implementation of both pacemaker and heart.

Project Highlights – Simulator

- Developed A CMSIS OS simulator for Windows
  - (OS Used by MBED devices)
  - Rapid development and testing on a PC before deployment to MBED,
  - **Does not require any code changes in the MBED program**
    - just compiling it using a PC compiler and linking with a stub library.
  - Allows for comprehensive testing using tools available for PC
    - Automatic testing, code analysis, code coverage tools, etc',
  - Supports multiple MBED programs running concurrently,
  - Simulates multiple hardware devices/sensors, and supports interconnects between the simulated MBED platforms,
  - Supports visualization/debug plugins

A-A timing

- "In the early days of dual-chamber pacing, all devices used what is commonly called ventricular-based timing. .... . In the mid-1980s, pacemaker manufacturers decided to fix this timing quirk by changing to what is now called atrial-bases timing. In atrial-based timing, it is an atrial event (senses or paced) that starts the interval knows as the AA interval."
  - "The Nuts and Bolts of Cardiac Pacing", 2nd Edition, by *Tom Kenny*, in Chapter 10 (page 86)

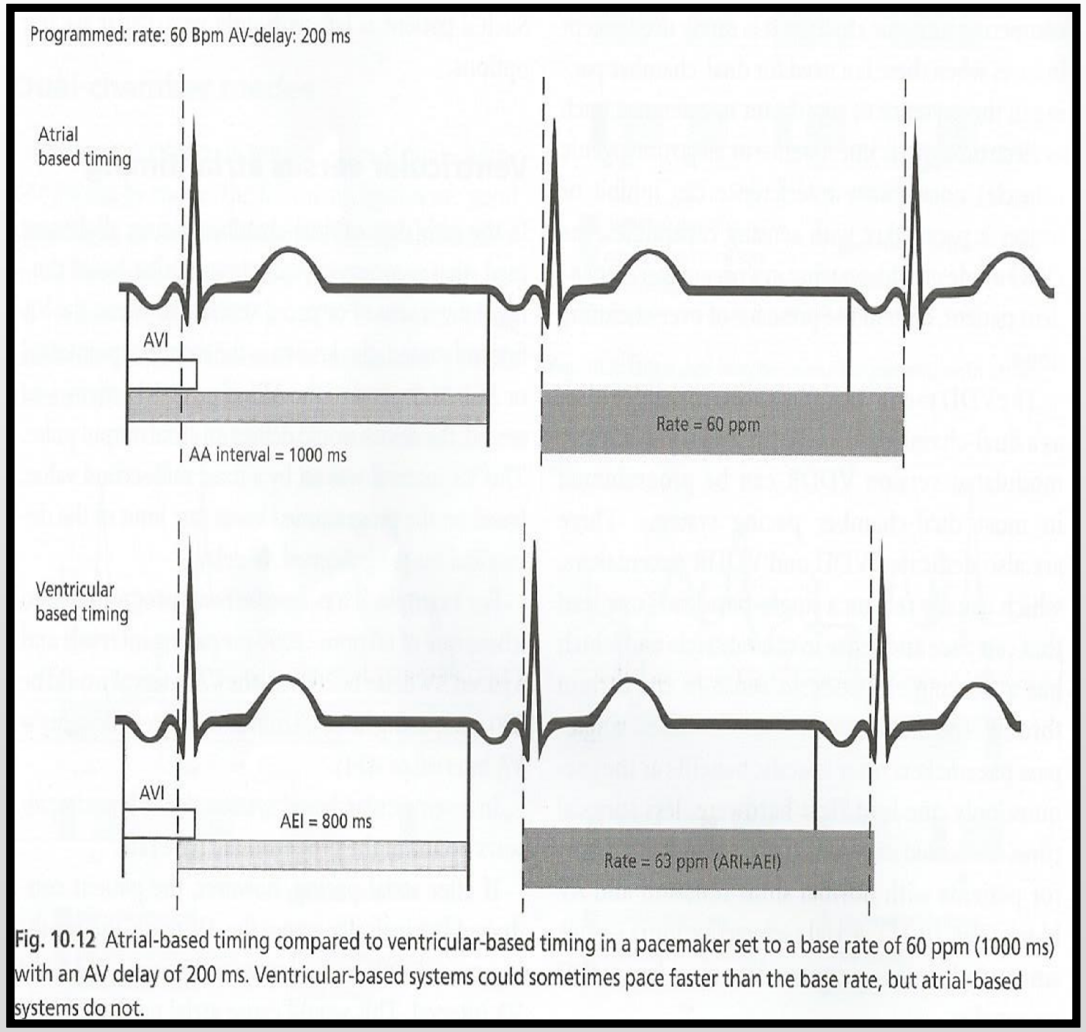
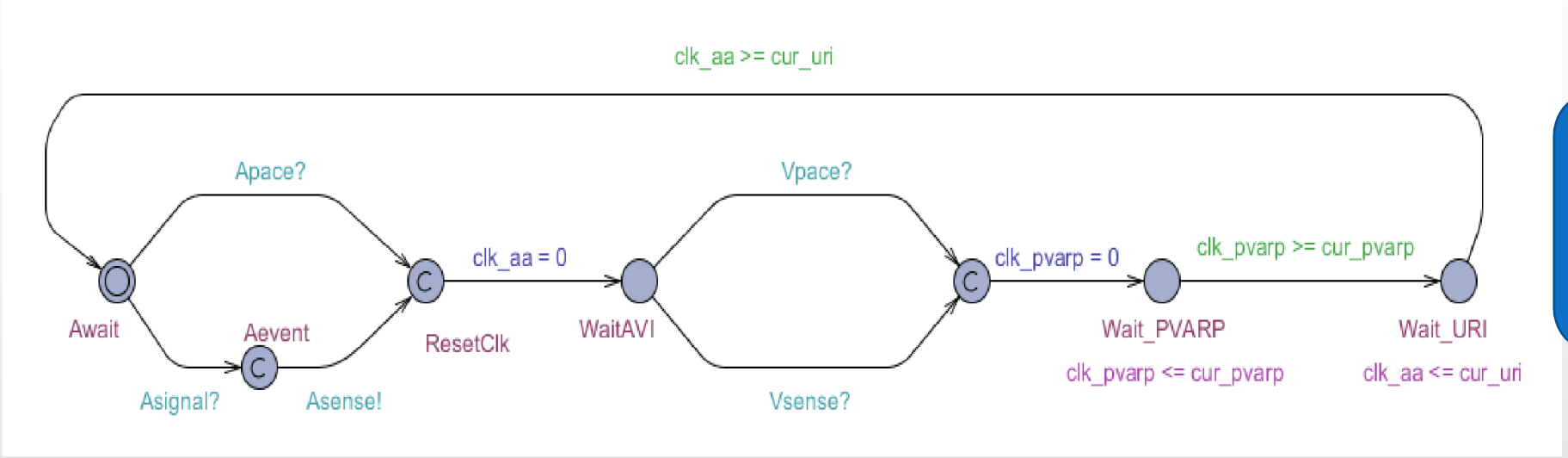
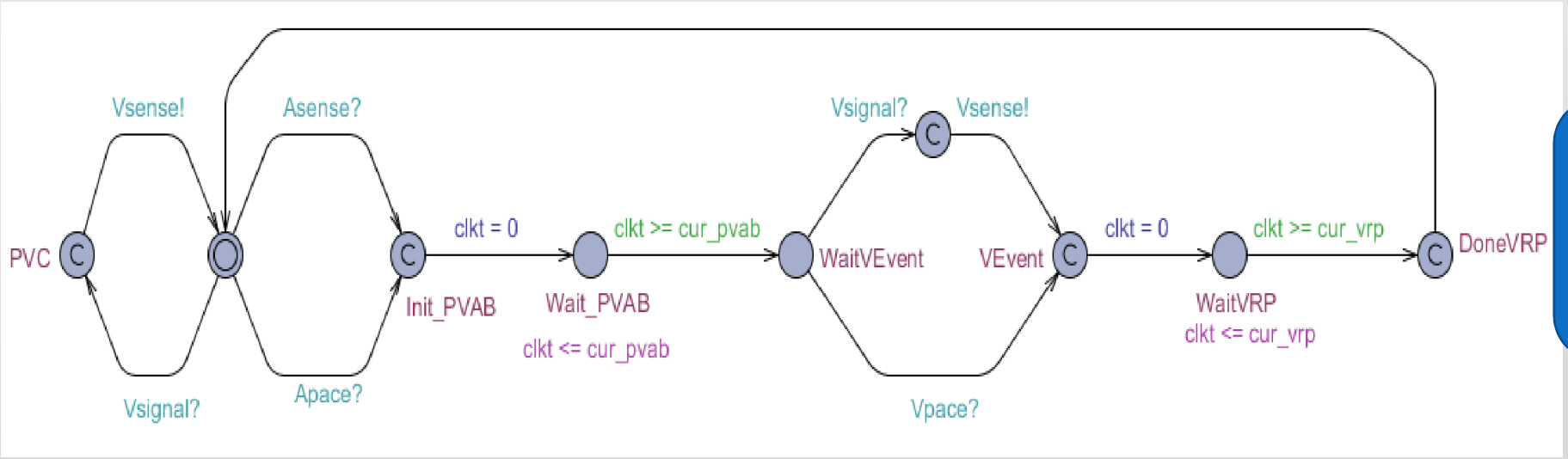


Fig. 10.12 Atrial-based timing compared to ventricular-based timing in a pacemaker set to a base rate of 60 ppm (1000 ms) with an AV delay of 200 ms. Ventricular-based systems could sometimes pace faster than the base rate, but atrial-based systems do not.

Modelling – Sensing

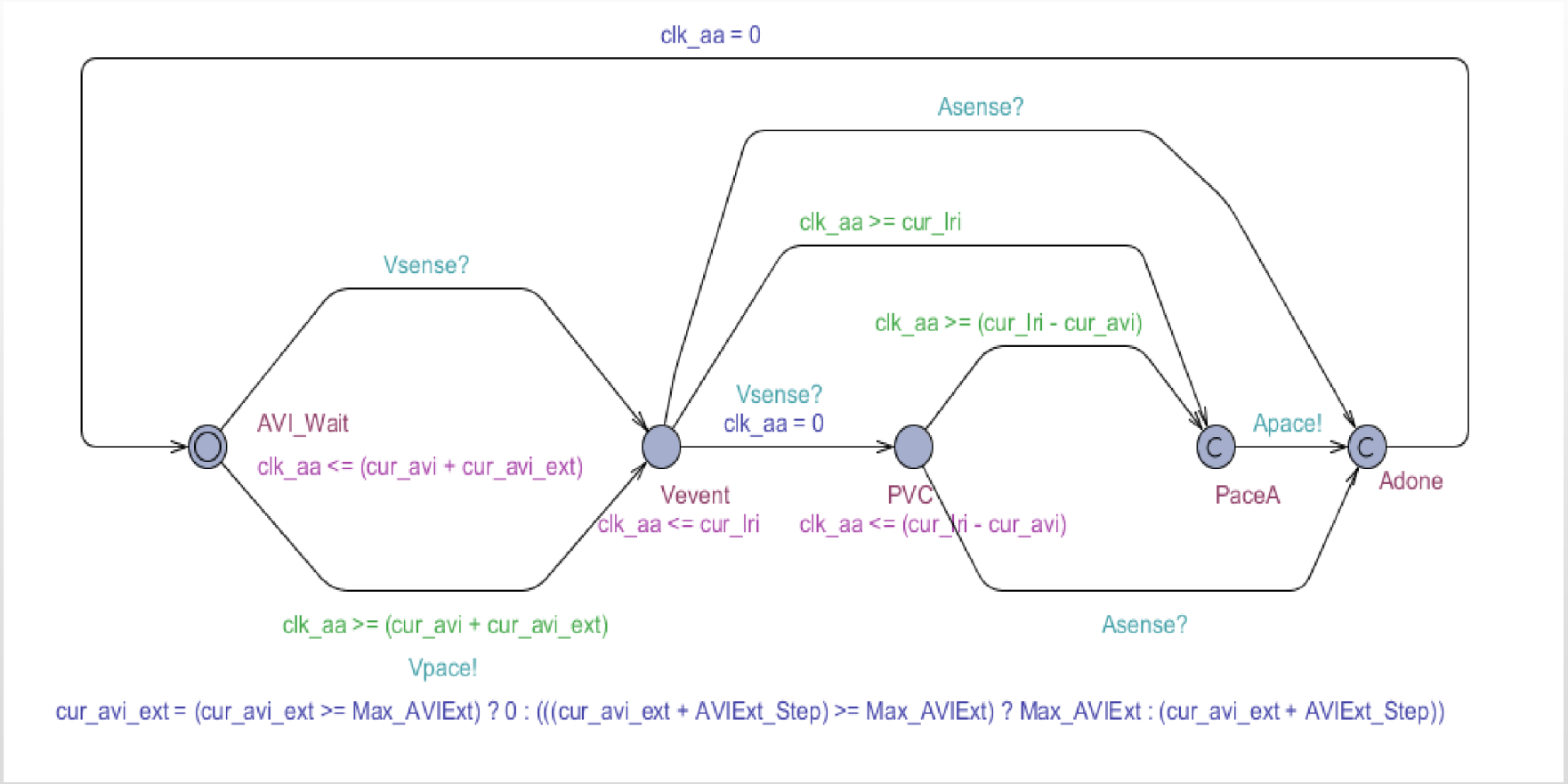


Atrial

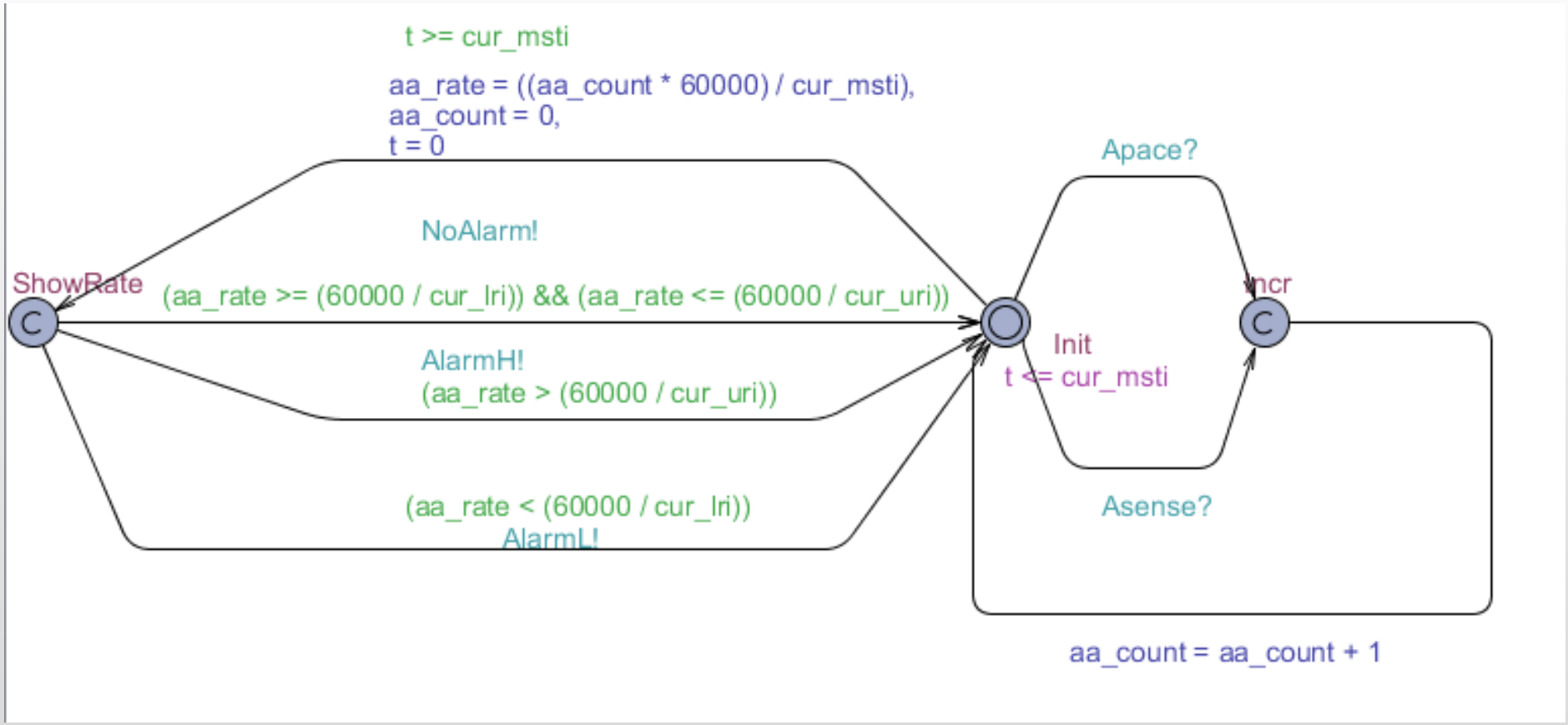


Ventricular

Modelling – Pacing



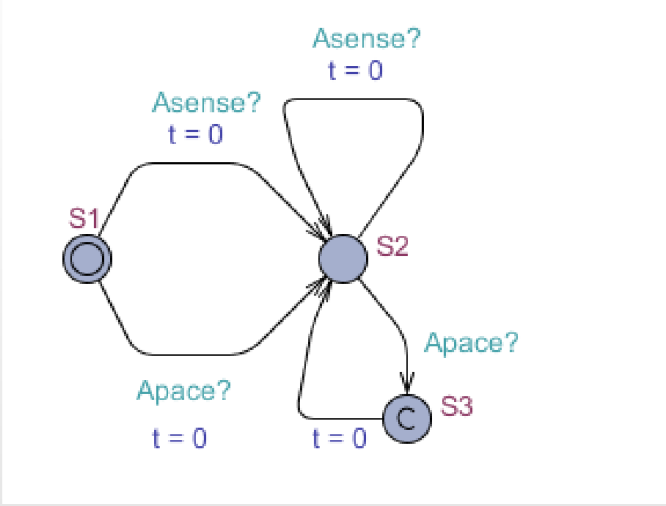
Modelling – Rate & Alarm

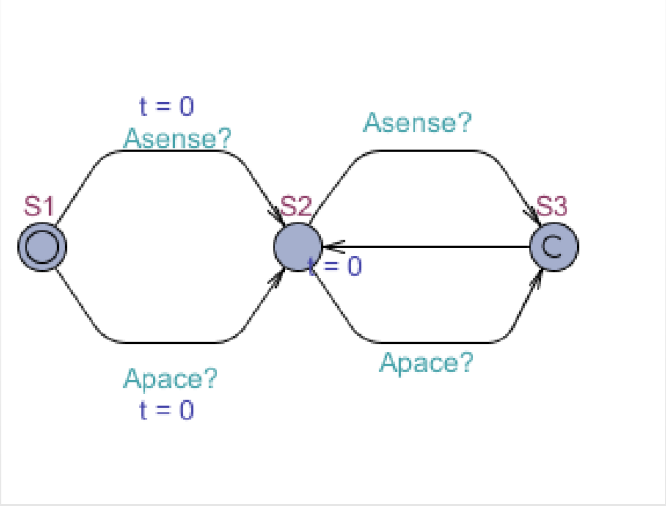


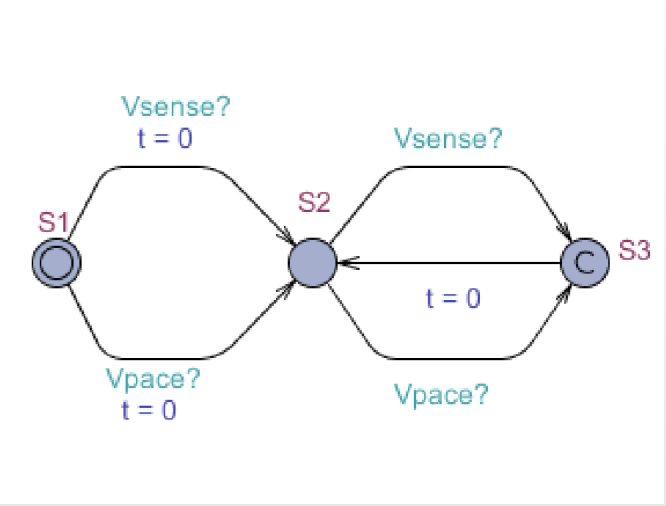
Verification - Basic

- No Deadlock
  - A[] !deadlock
- LEDs
  - A<> PM_Pacer_AA.PaceA imply PM_Leds_Apace.On
- More interesting verification requires helper “observer” process

Verification with Observers







AVI, URI, PVARP

A[] Observer_URI.S3 imply Observer_URI.t >= (cur_avi + cur_avi_ext + cur_pvarp)

...

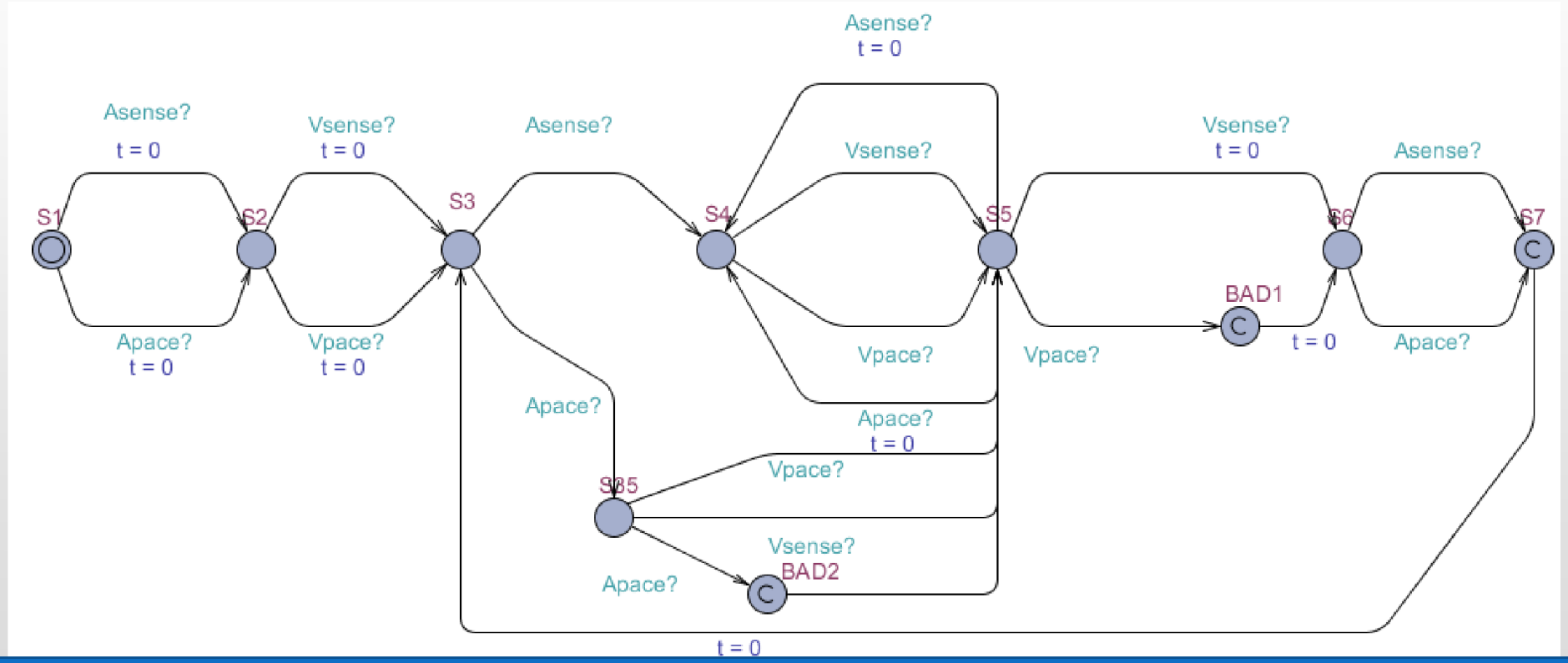
LRI

A[] Observer_LRI.S3 imply Observer_LRI.t <= cur_lri

VRP

A[] Observer_VRP.S3 imply Observer_VRP.t >= cur_vrp

Verification with Observers



BAD1 - The pacemaker will never generate two ventricular paces without an atrial event in-between
BAD2 - The pacemaker will never generate two atrial paces without an ventricular event in-between

Code Generation

Setup Code

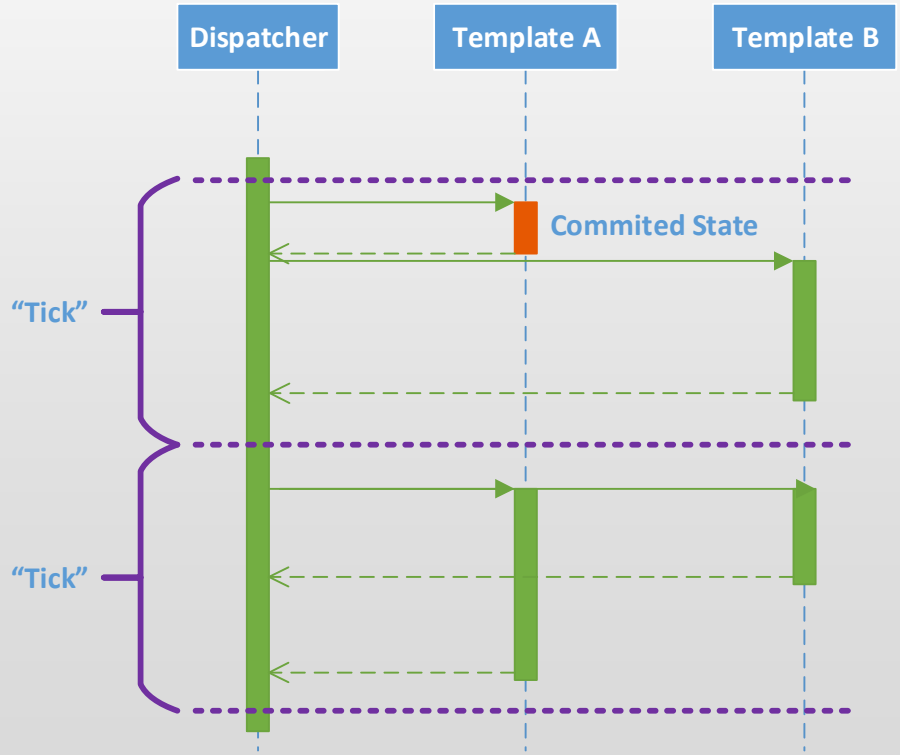
UPPAAL Model

Autogen'd Code

MBED binary

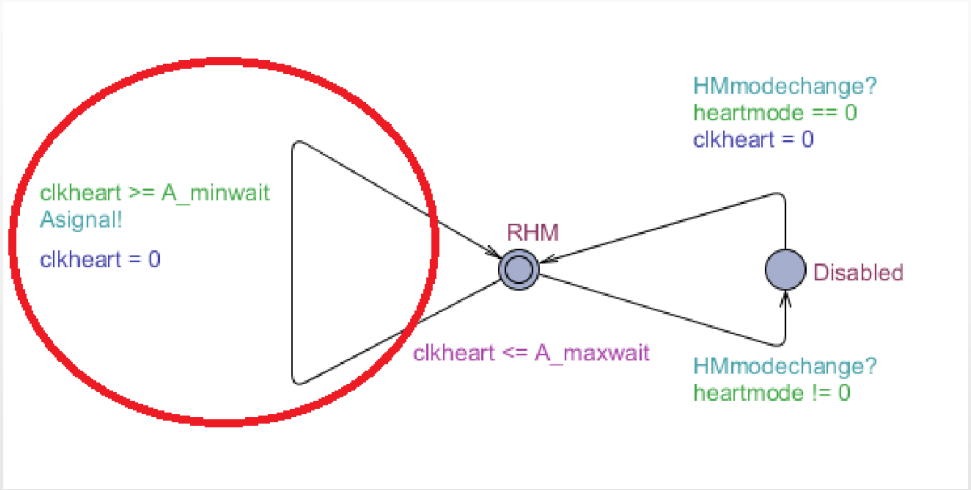
Code Generation

- Main challenge was handling committed/urgent/regular semantics as accurately as possible



Code Generation – Invariants, Guards and Updates

- Generator fully supports invariants, guards and updates
  - Generate function to calculate them at runtime
- Variables (int, bool, clocks, ...) stored in a global object




```
static bool guardfunc_HM_Asignal_TRANS_RHM__RHM__0(){
    return ((((int)SystemVariables.clk_GLOBAL_clkheart) >=
                ((int)SystemVariables.int_GLOBAL_A_minwait)));
};
static void updatefunc_HM_Asignal_TRANS_RHM__RHM__0() {
    SystemVariables.clk_GLOBAL_clkheart = (0);
};
```

Code Generation - Integration with Hardware

```
void set_channel_action(SYNCHRONIZATION_CHANNEL* channel,
                        PinName pin,
                        SendChannelMode mode,
                        int pulse_length_us = 0);
```

SendChannelModeNone	Do nothing
SendChannelModeSet	Set the pin to 1
SendChannelModeToggle	Toggle the value of the pin
SendChannelModeReset	Set the pin to 0
SendChannelModePulseUp	Pulse up to 1 for the time specified, then reset to 0
SendChannelModePulseDown	Pulse down to 0 for the time specified, then reset to 1

Code Generation - Integration with Hardware

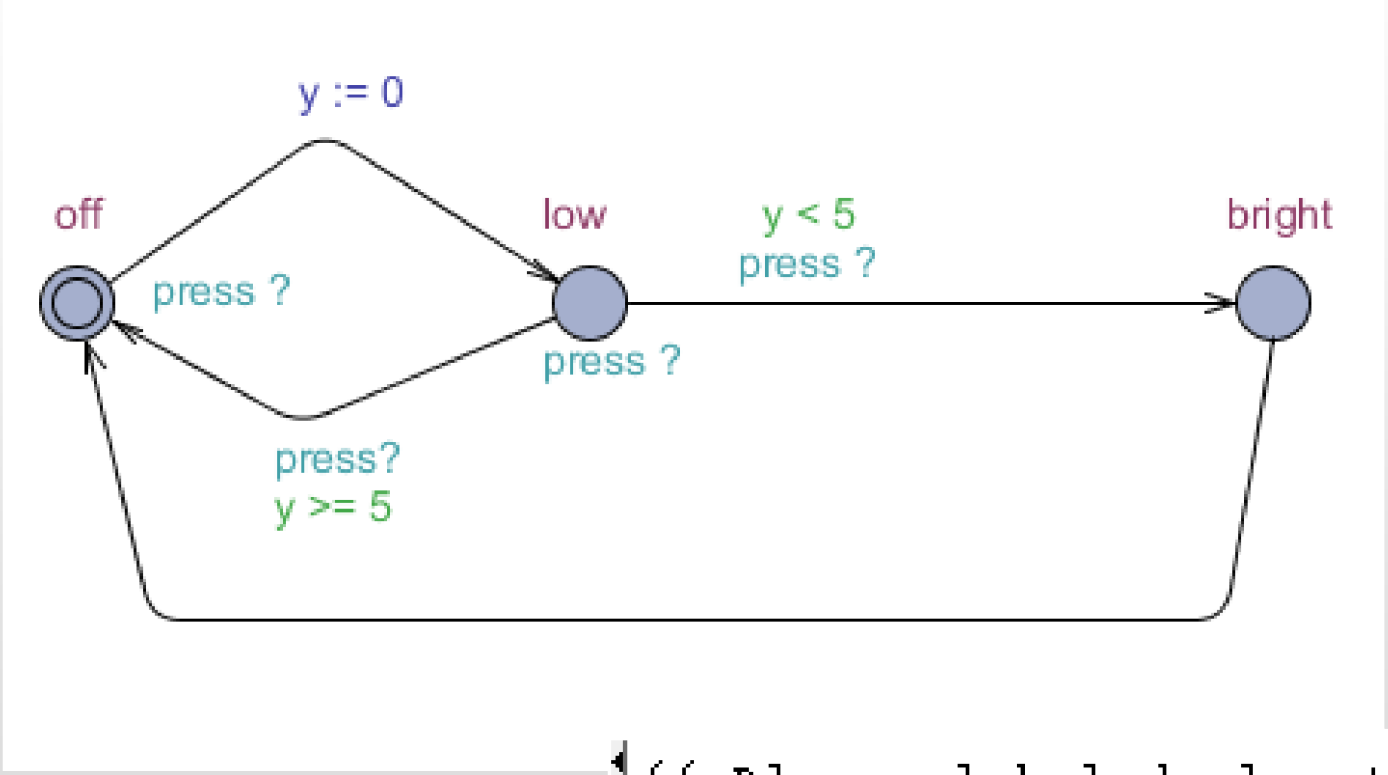
```
void set_receive_input(SYNCHRONIZATION_CHANNEL* channel,
                       PinName pin,
                       PinMode mode, // pull up, pull down, ...
                       ReceiveChannelMode rcmode)
```

ReceiveChannelModeNone	Do nothing
ReceiveChannelInterruptRise	Fire the channel on rise
ReceiveChannelInterruptFall	Fire the channel on fall

Code Generation - Callbacks

- All operations support adding user callbacks
  - Enter state
  - Leave state
  - Take transition
  - State template
  - ...
- Callbacks can do anything but should be fast (like ISRs)

Code Generation – Example
Light with Two Levels




```
// Place global declarations here.
chan press;
clock y;
```

Code Generation – Example
Light with Two Levels

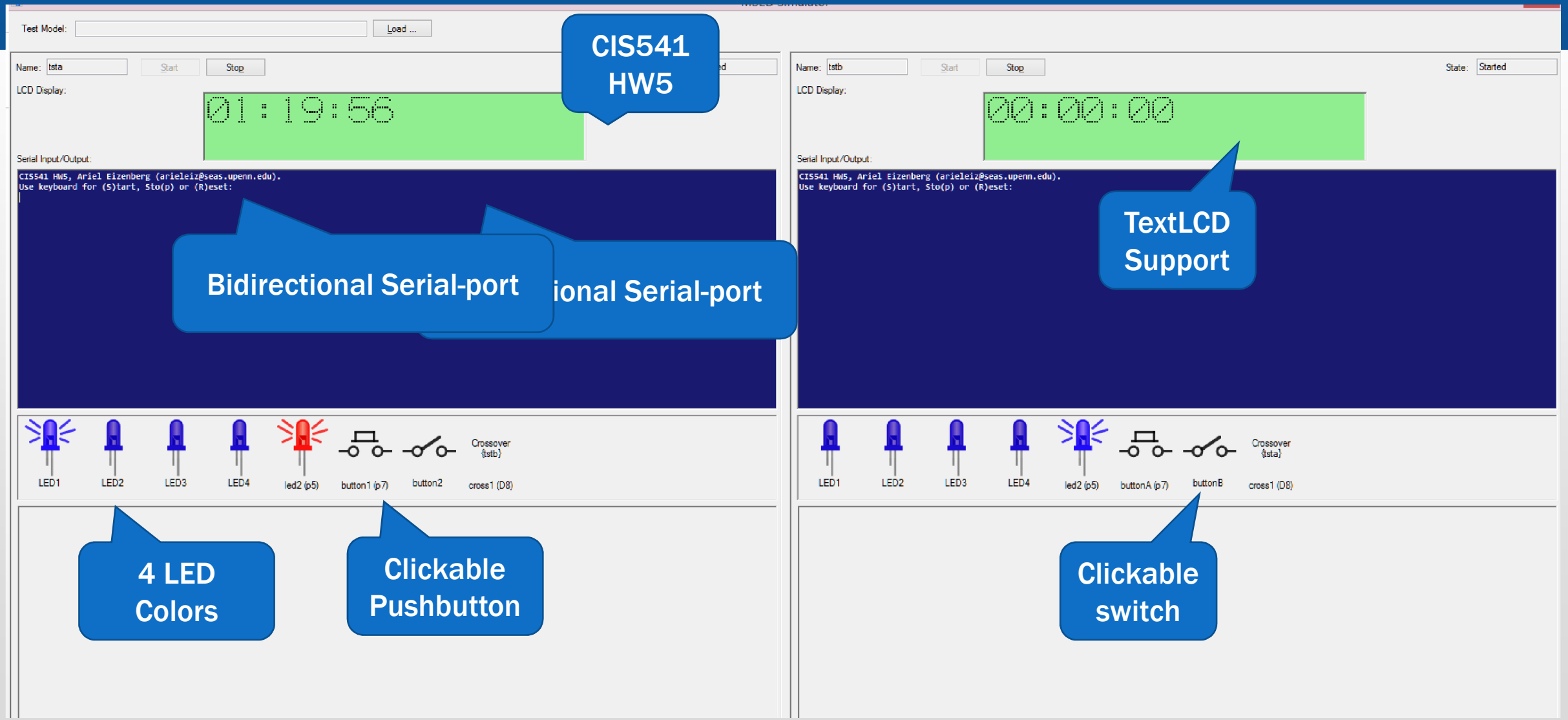
Emulates two levels with two LEDs

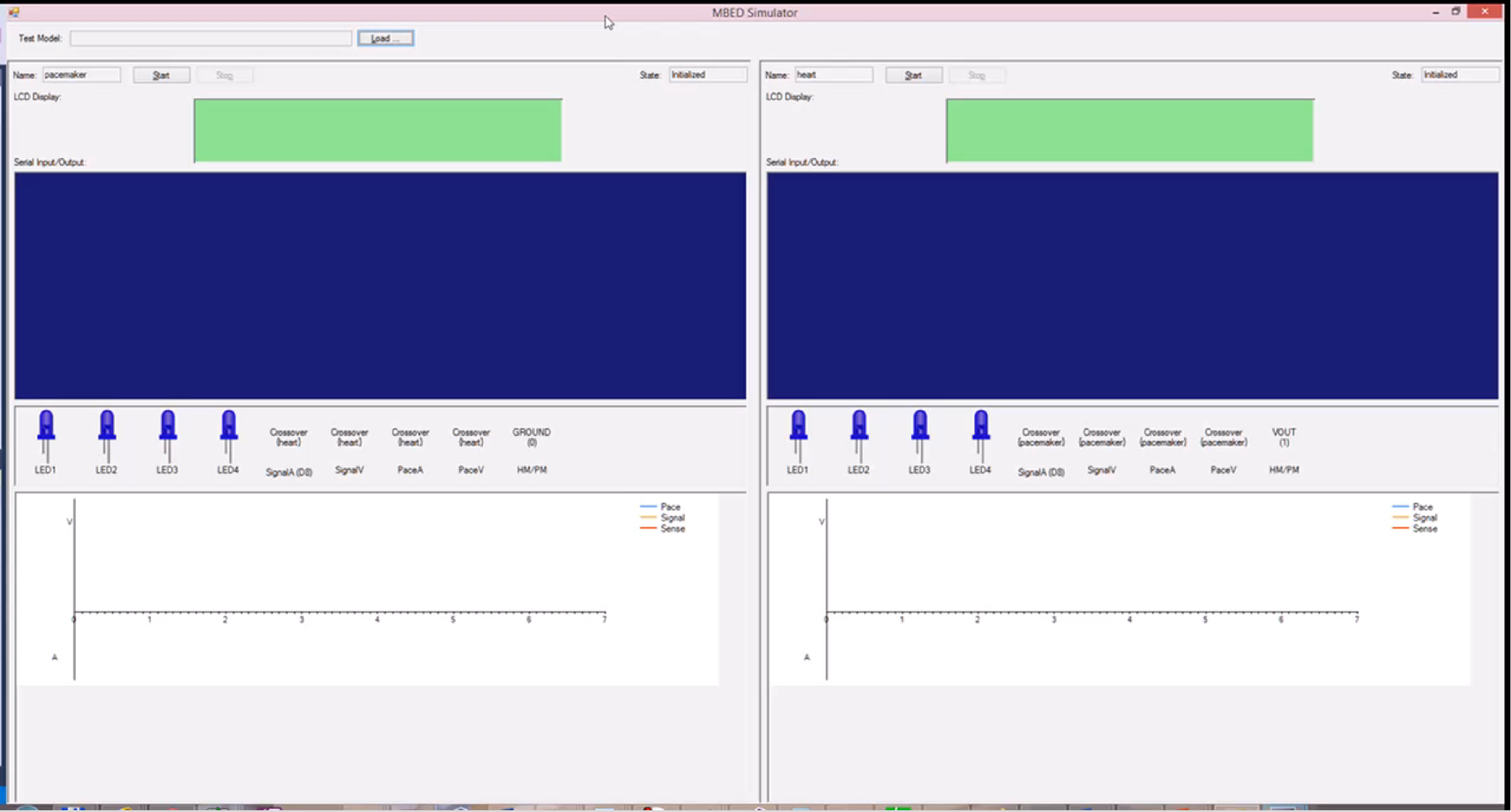
```
int main (void) {
        Dispatcher d;
        d.set_clock_multiplier(1000000 /*us*/); // a tick each second
        d.set_channel_action(&Light_TRANS_off_low_0, LED1, SendChannelModeSet);
        d.set_channel_action(&Light_TRANS_low_off_0, LED1, SendChannelModeReset);
        d.set_channel_action(&Light_TRANS_low_bright_0, LED2, SendChannelModeSet);
        d.set_channel_action(&Light_TRANS_bright_off_0, LED1, SendChannelModeReset);
        d.set_channel_action(&Light_TRANS_bright_off_0, LED2, SendChannelModeReset);
        disp.add_process(&Model::Light_PROCESS_STATE);
        disp.run();
}
```

MBED Simulator

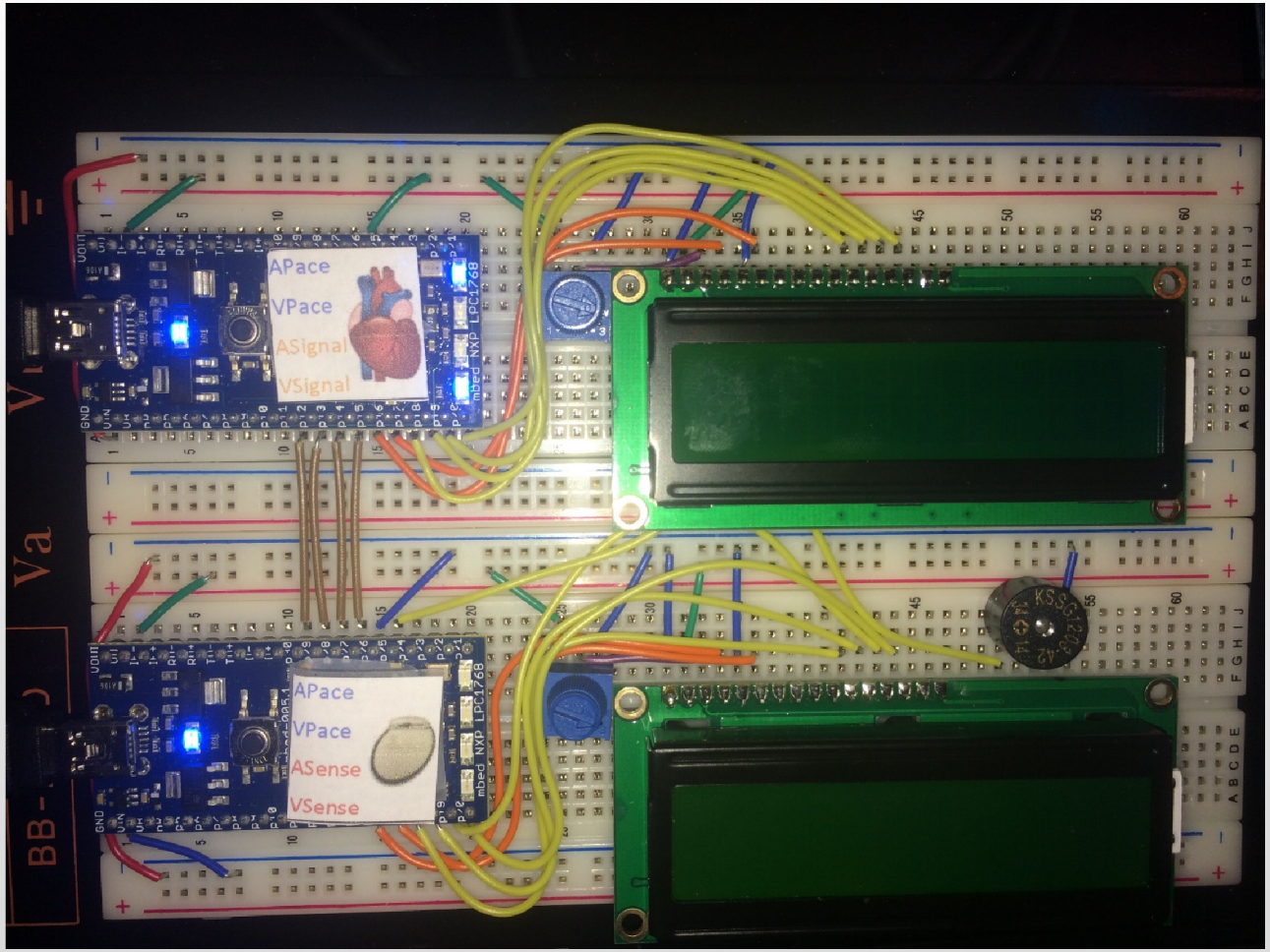
- MBED development environment is early 1990s style
- Does not require any change to code, just compile & link with library
  - Develop in modern IDE
  - PC Debugging tools, Code Analysis, Code Coverage, …
- Supports:
  - Threads, including RTOS style signaling
  - Timers, Tickers, RTOSTimer, etc' – reading and interrupts,
  - Mutexes,
  - GPIO read, write, interrupts,
  - TextLCD,
  - Serial Port – reading, writing, interrupts, break,
  - Sleep,
  - NVIC_SystemReset.

MBED Simulator




MBED Simulator
Test Model:
Load ...
Name: pacemaker
Start
Stop
State: Initialized
LCD Display:
Serial Input/Output:
LED1
LED2
LED3
LED4
Crossover (heart)
SignalA (D8)
Crossover (heart)
SignalV
Crossover (heart)
PaceA
Crossover (heart)
PaceV
GROUND (0)
HM/PM
Pace
Signal
Sense
V
A
0
1
2
3
4
5
6
7
Name: heart
Start
Stop
State: Initialized
LCD Display:
Serial Input/Output:
LED1
LED2
LED3
LED4
Crossover (pacemaker)
SignalA (D8)
Crossover (pacemaker)
SignalV
Crossover (pacemaker)
PaceA
Crossover (pacemaker)
PaceV
VOUT (1)
HM/PM
Pace
Signal
Sense
V
A
0
1
2
3
4
5
6
7

The Hardware



Debugging on actual MBED

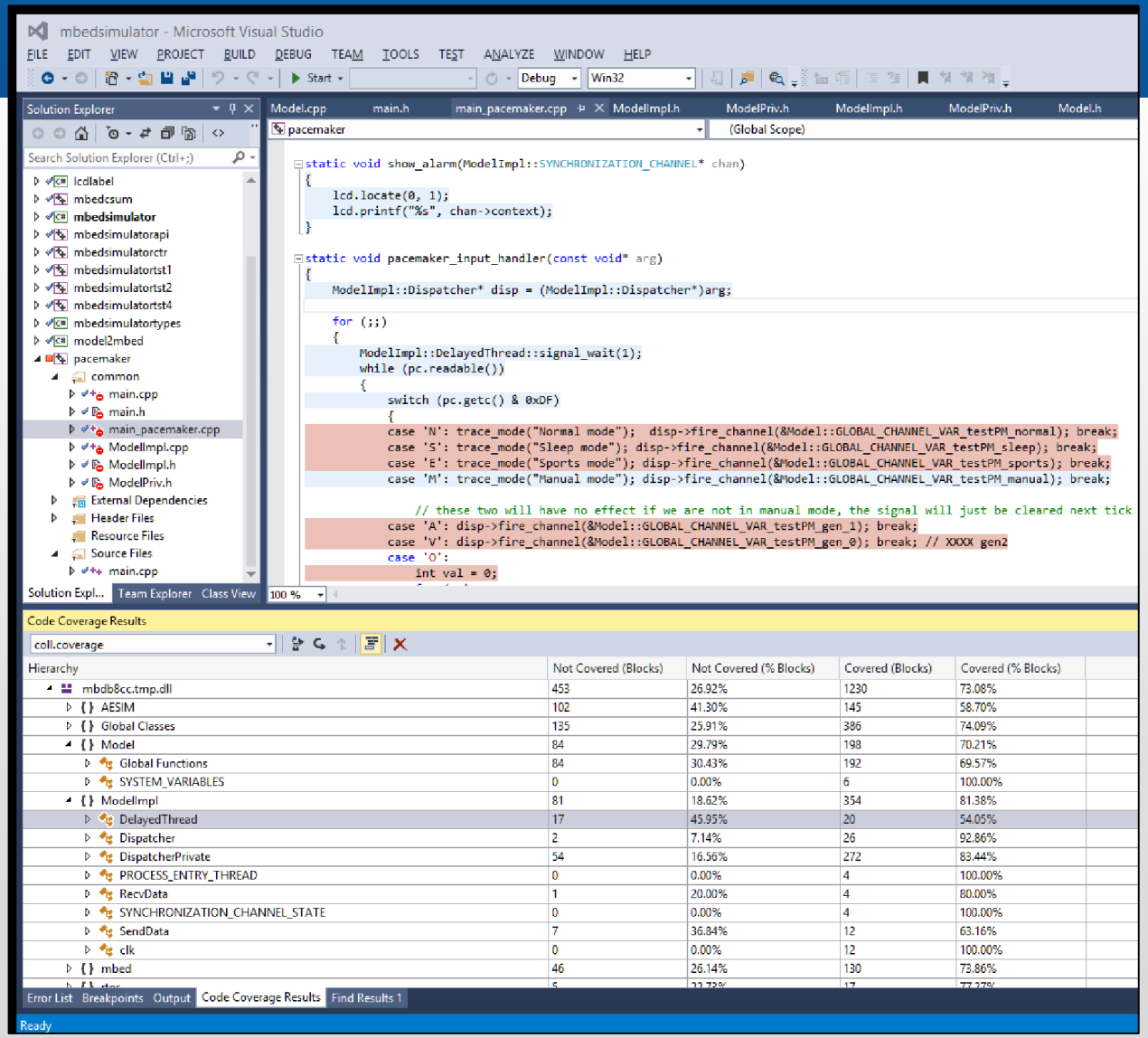
- MBED IDE does not support debugging …
- And I have no JTAG connector, so …
- Build project using GCC-ARM-EMBEDDED 4.8
  - Generates bad code with -Os, so no optimizing
- Debug using gdb+PyOCD

Testing

- Developed Serial logger + visualizer
  - Automatically parses serial output from both devices
  - Add timing information to logs
  - Provide a quick way to see if everything is OK

Testing – Code Coverage

- Use simulator in conjunction with PC Code Coverage Tools



Assurance Case