Coordination of Synchronous Components with Time Triggered Programming

Department of Computer Science and Engineering
IIT Bombay

Under the guidance of

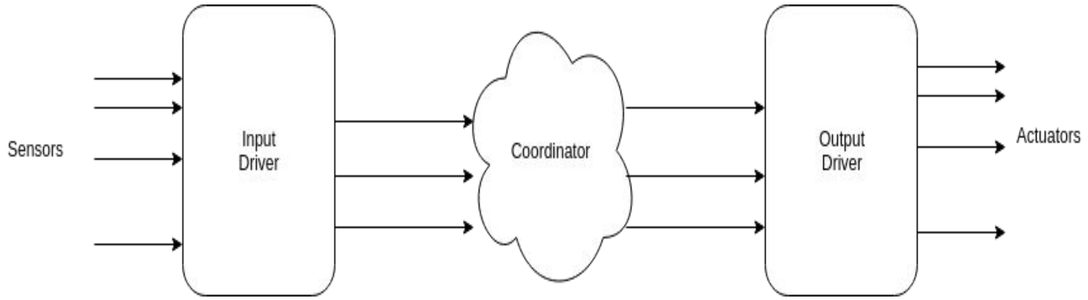
Prof. Paritosh K. Pandya and
Prof. Kavi Arya

Presented by

Avais Ahmad

Big Picture

- Synchronous languages lustre/Heptagon is good for **coordination** of embedded systems



- But they follow **Synchrony Hypothesis:-**
  - Computation and communication takes negligible time
- So it fails for **time intensive** task

Big Picture

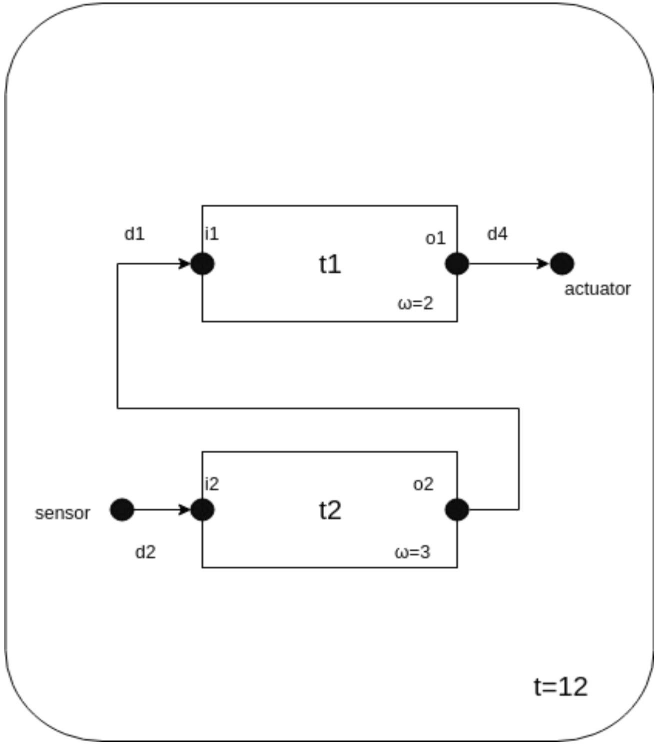
- To solve this we combined **Heptagon components** with a **Time Triggered Language** TTProg inspired by Giotto
    - Thomas A. Henzinger, Benjamin Horowitz and Christoph M. Kirsch,"**Giotto**: A Time-triggered Language for Embedded Programming", in Proceedings of the IEEE, vol. 91. IEEE, 2003

    - **Heptagon** language: Gwenaël Delaval, Hervé Marchand, Marc Pouzet, Eric Rutten

- We call this **High level language** as **TTProg[Heptagon],** this allow components whose execution takes time
- We retain some of advantages of both

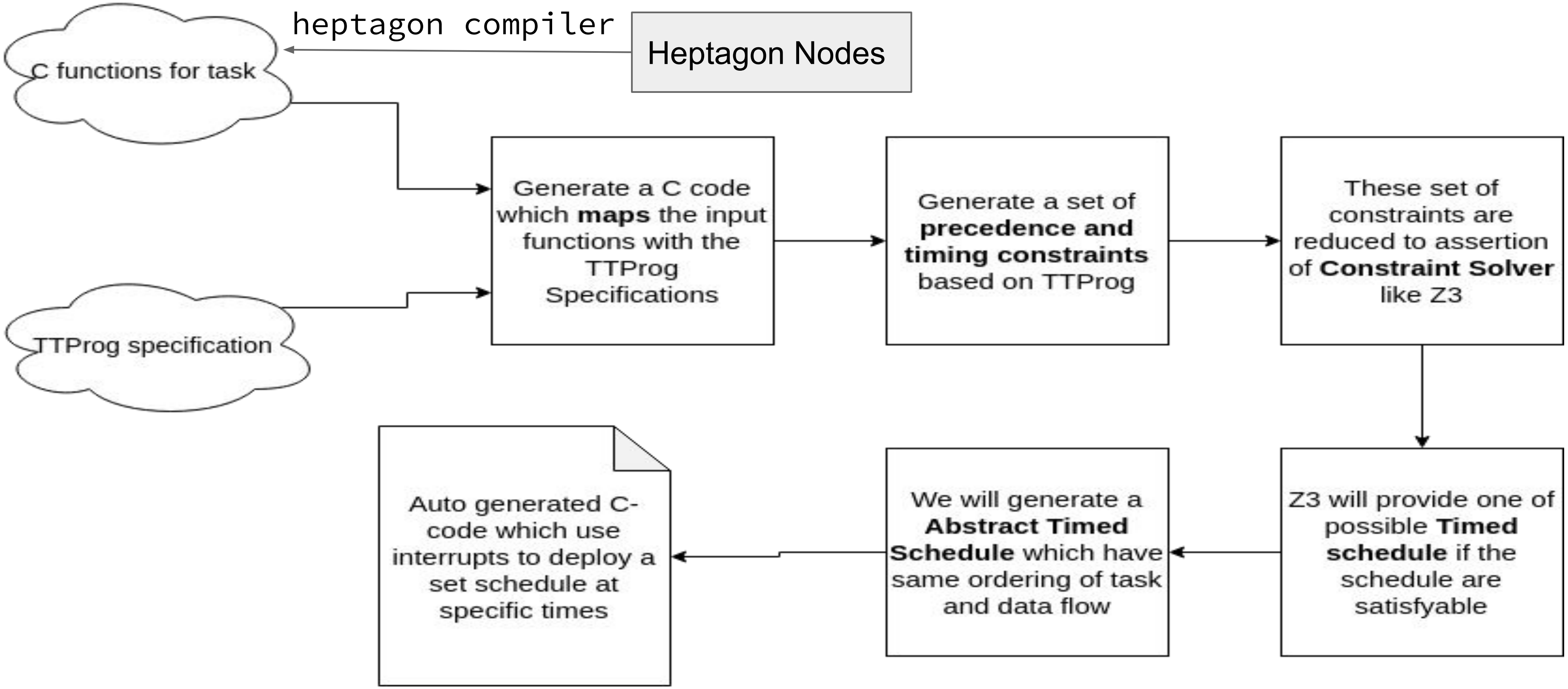
Big Picture

- The components will be combined in deterministic fashion and may have data dependencies
- It will also have a set of timing constraints



What we did

We built a multiphase automated compiler which does the following:-



Motivation

- We implement Static Scheduling
- It is a bare metal implementation
- Does not require any RTOS or Context Switching
- Deterministic and memory will be statically initialized
- Very efficiently executable code
- It will give extreme speed to tight application

Literature Surveys

Synchronous Programming Languages

- State Charts
- Lustre
- Heptagon

Asynchronous Programming Languages

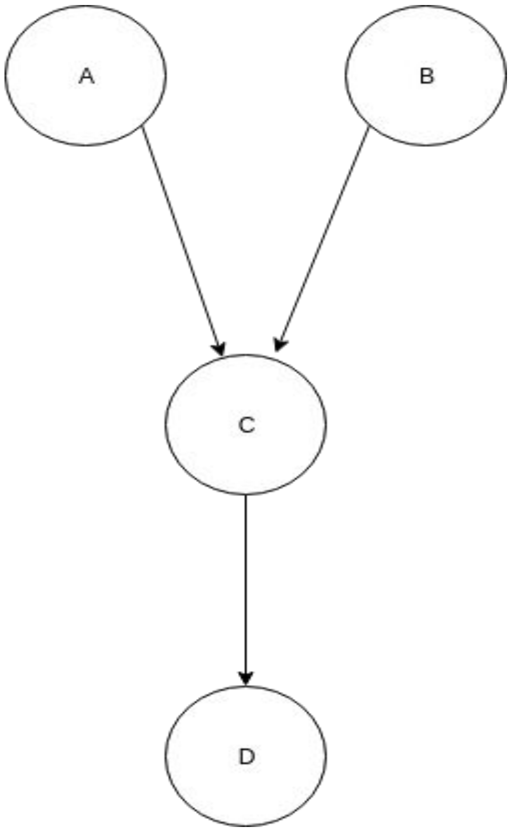
- Linda

Time Triggered Programming Languages

- Giotto

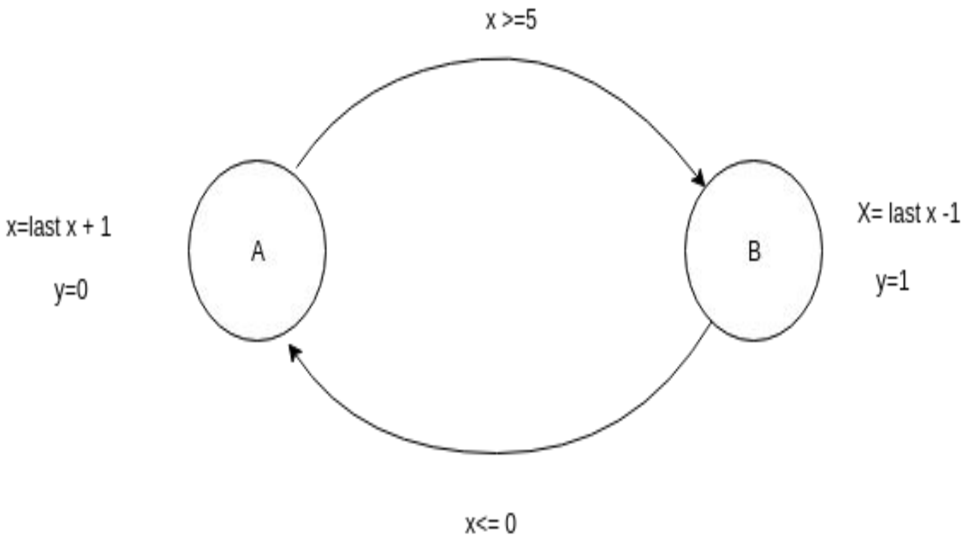
Heptagon

- It is a synchronous dataflow language
- Declarative language
- Very useful in designing reactive systems
- For each step cycle the node will read the value from input stream and write the value as output stream

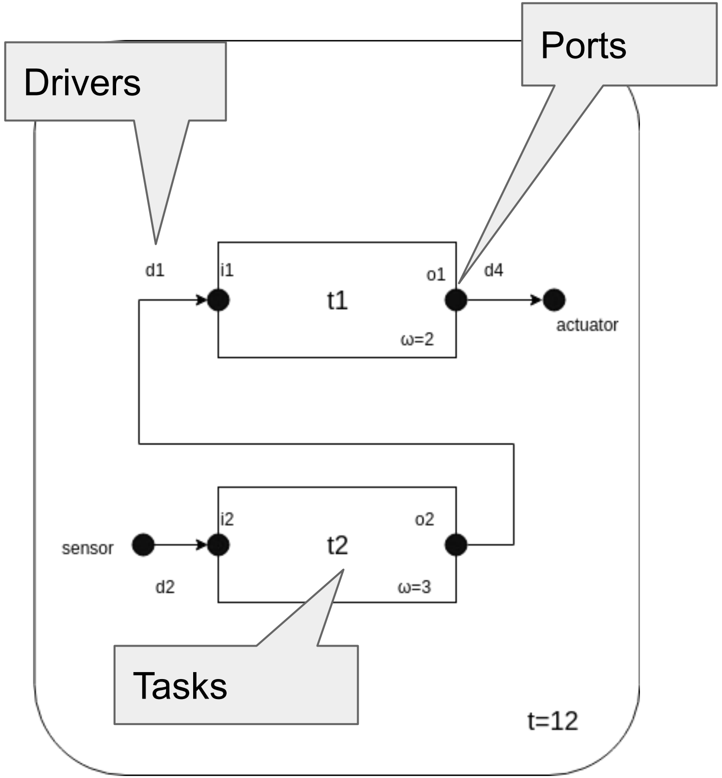


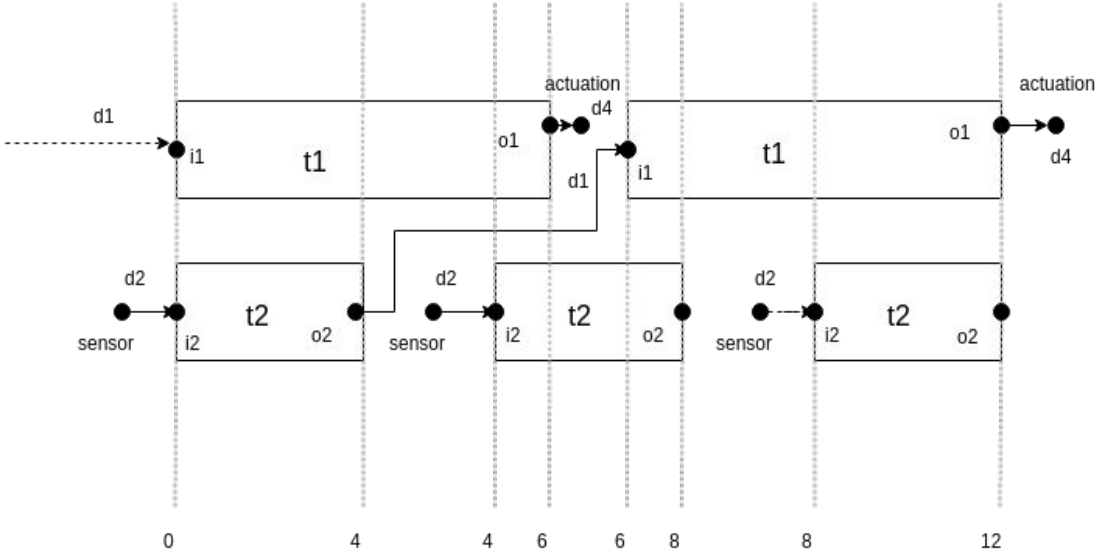
Heptagon Automata

```
node atm() returns(z,y:int)
var last x: int =0;
let
    z=x;
    automaton
        state A
            do x= last x +1; y=0
            until x>=5 then B
        state B
            do x= last x -1;y=1
            until x<=0 then A
    end
tel
```



TTProg Inspired by Giotto

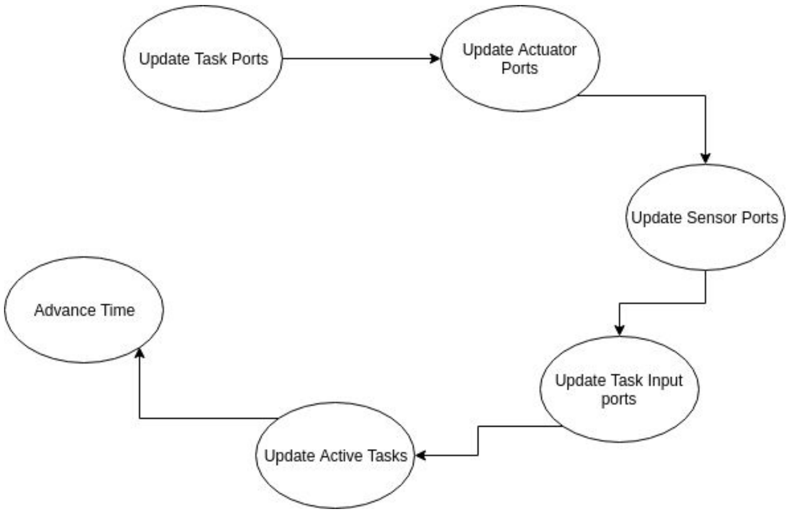


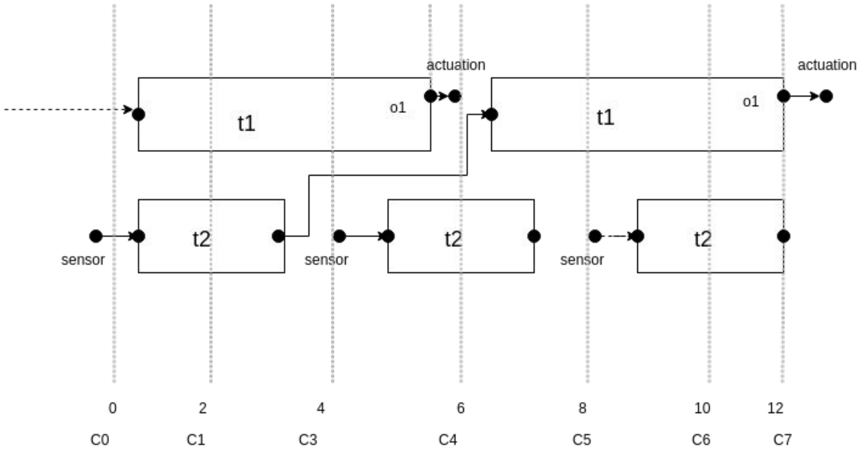


TTProg Configuration and Micro Steps

$$C = (m, \delta, v, \sigma_{active}, \tau)$$

$$C_{update} = \frac{T}{\omega_{LCM}}$$

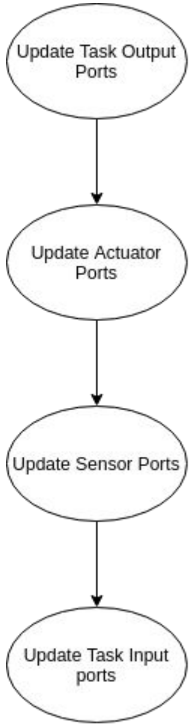




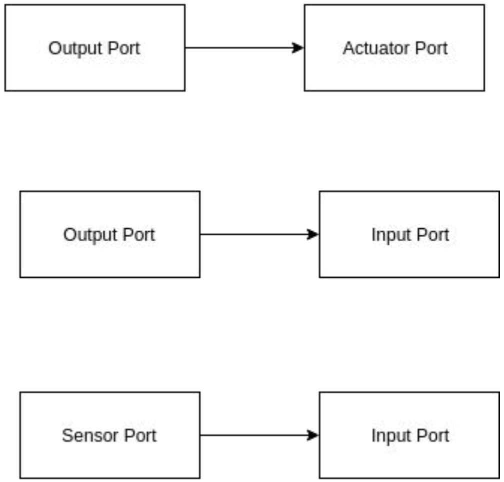
TTProg[Heptagon]

Giotto Tasks		Heptagon Nodes
task name (t)	⟺	node name
Input Ports	⟺	node inputs
Output Ports	⟺	node outputs
function (f)	⟺	step function

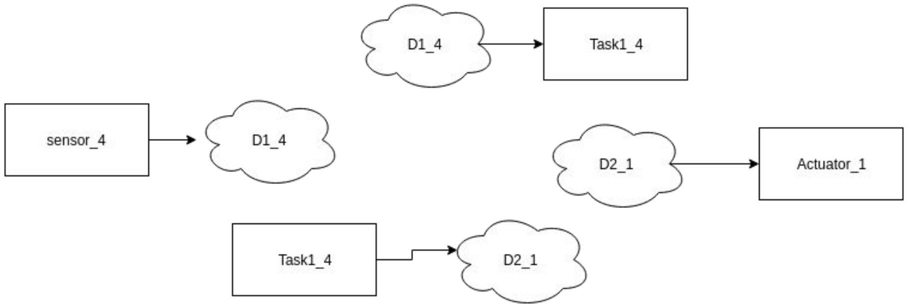
TTProg Data Flow



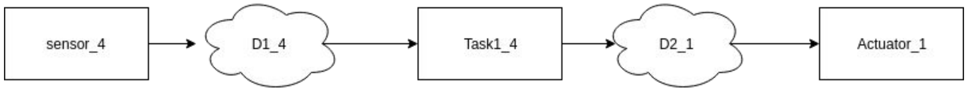
Giotto Micro Steps



Micro Steps Dependencies

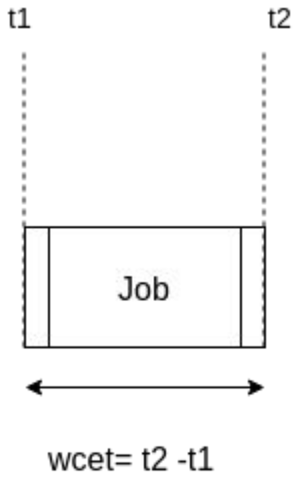


Individual Constraints



Data Flow

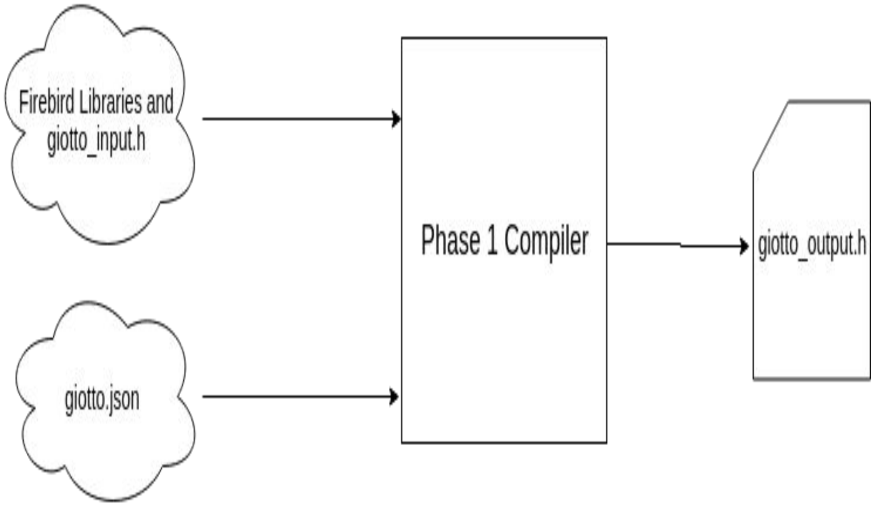
WCET



micros(): Number of microseconds since current execution of program

```
t1=micros();
f1(1);
t2=micros();
time=(t2-t1);
```

Phase 1 compiler



- All ports as global variable
- void functions for all the task generated and mapped with input functions
- Port values passed to these functions as specified
- Output of function is obtained by call by reference

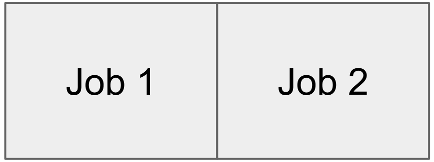
Phase 2 compiler-Serialization Constraints

$job_1, job_2 \in \{Drivers, Task\}$

$job_1 != job_2$

Constraints will be of form:

$job_1 + wcet_1 <= job_2 || job_2 + wcet_2 <= job_1$



Phase 2 compiler-Timing Constraints

Actuator

Lower Bound:

$$actuation_i >= \frac{ModePeriod}{ActuatorFrequency} * i - jitter$$

Upper Bound:

$$actuation_i + wcet <= \frac{ModePeriod}{ActuatorFrequency} * i$$

Sensor

Lower Bound:

$$input\_tn_i >= \frac{ModePeriod}{TaskNFrequency} * i$$

Upper Bound:

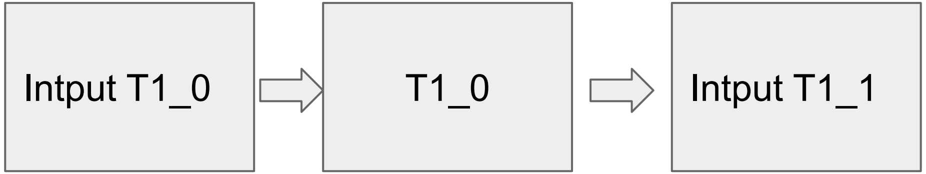
$$input\_tn_i + wcet <= \frac{ModePeriod}{TaskNFrequency} * i + jitter$$

Phase 2 compiler-Precedence Constraints

Driver-Task

$$input\_tn_i + wcet <= tn_i \text{ , where wcet is for the driver}$$

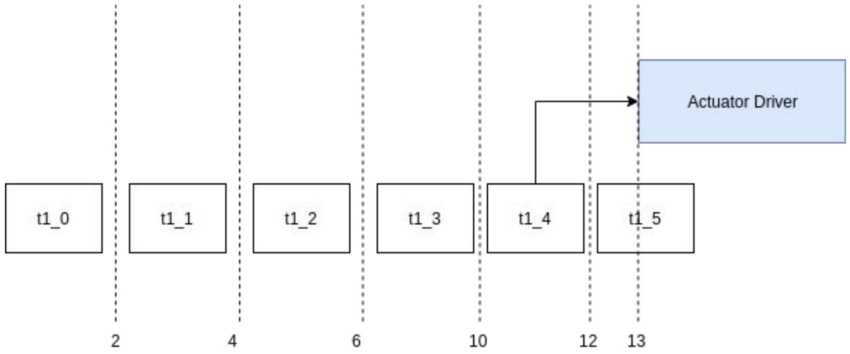
$$tn_i + wcet <= input\_tn_{i+1} \text{ , where wcet is for the task}$$



Phase 2 compiler-Precedence Constraints

Task-Actuator

$$j = \frac{LogicalStartTimeOfActuation_i}{LogicalTimePeriodOft1}$$



Actuation i occur after task instance j

$$t1_j + wcet_{task} <= actuation_i$$

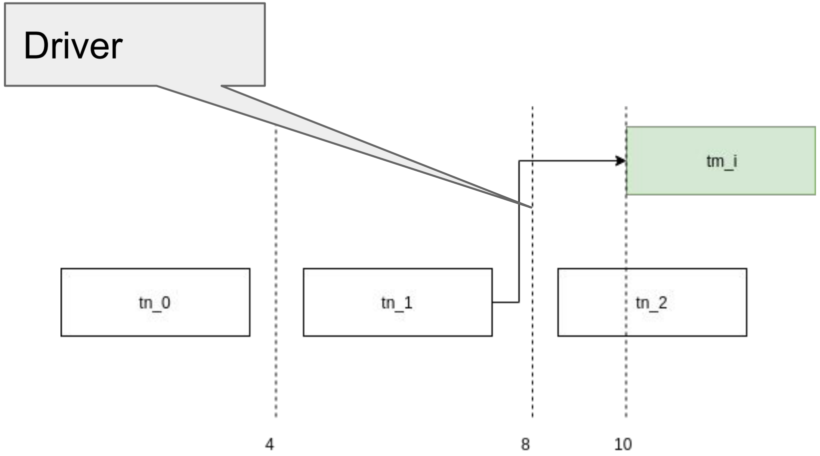
Task instance j+1 should occur after the actuation

$$actuation_i + wcet_{act} <= t1_{j+1}$$

Phase 2 compiler-Precedence Constraints

Task-Driver

$$j = \frac{LogicalStartTimeOft1_i}{LogicalTimePeriodOft2}$$



Instance i of t1 should occur after instance j of t2

$t2_j + wcet_{t2} <= input\_t1_i$

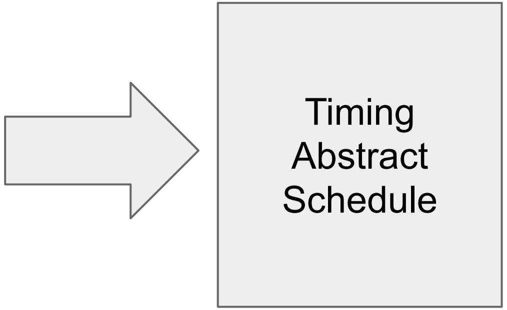
To make sure other instance of t2 run after input driver of $t1_i$

$input\_t1_i + wcet_{input\_t1} <= t2_{j+1}$

Phase 3 COmpiler

```
avais@avais-Lenovo-Z50-70:~/Documents/P
sat
[input_t2_2 = 8,
 input_t2_1 = 4,
 t1_update_0 = 9/20,
 t2_update_0 = 9/2,
 t1_1 = 19/4,
 t2_2 = 19/2,
 input_t1_0 = 0,
 actuation_1 = 11,
 input_t2_0 = 1/2,
 t1_0 = 1/4,
 t2_0 = 5/4,
 actuation_0 = 5,
 t2_update_1 = 37/4,
 t2_1 = 17/2,
 t1_update_1 = 6,
 t2_update_2 = 41/4,
 input_t1_1 = 1]
```

Timed Schedule



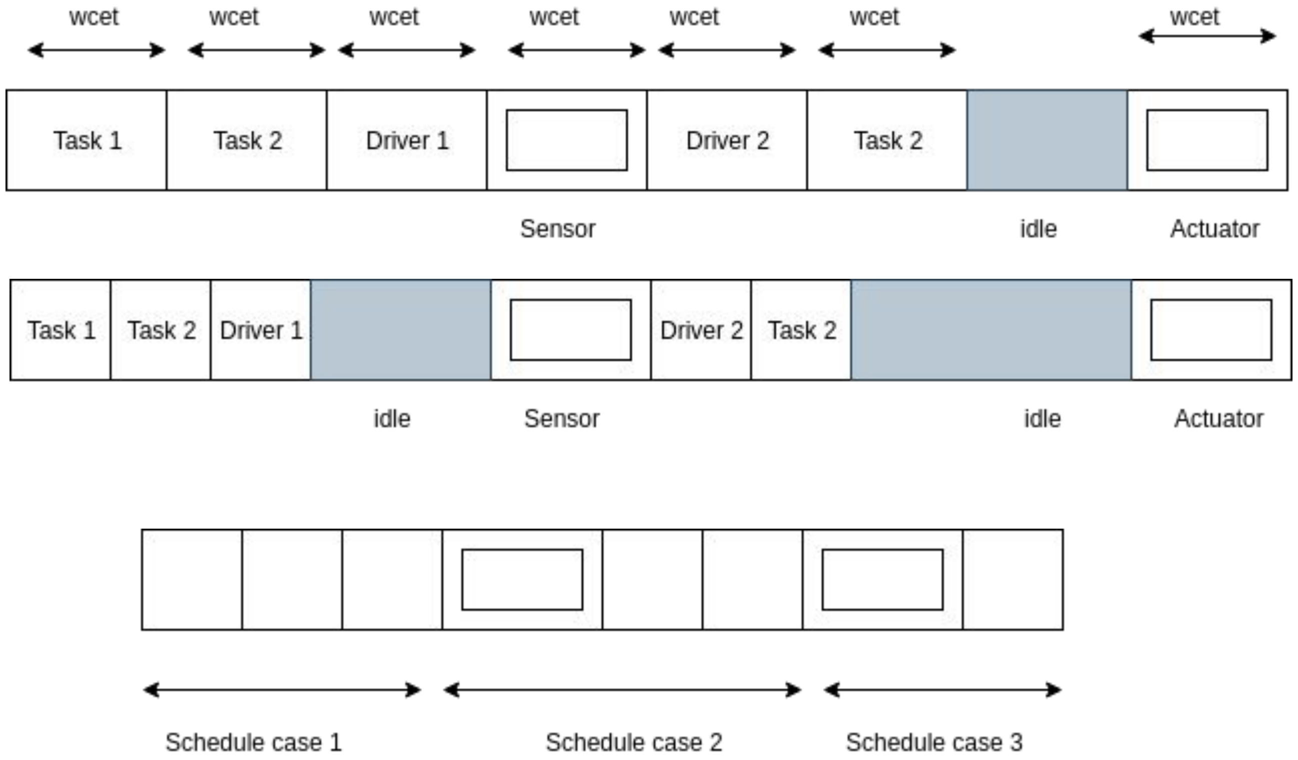

```
input_t1
t1
wait

input_t2=0.5
t2
input_t1
wait

input_t2=4
t2
wait

actuation
t1
wait

input_t1=8
t2
wait

actuation=11
```

Phase 3 COmpiler

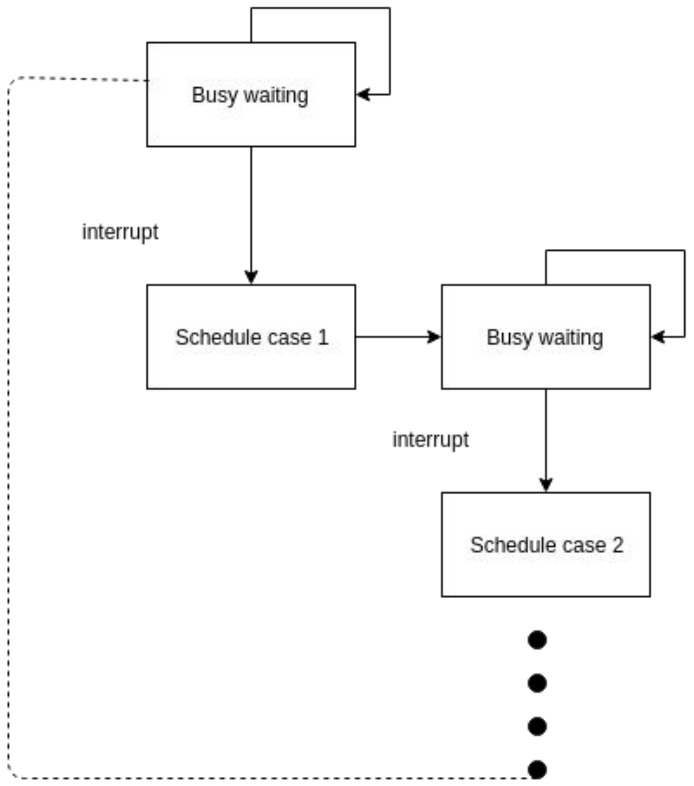
Initial Timed Schedule

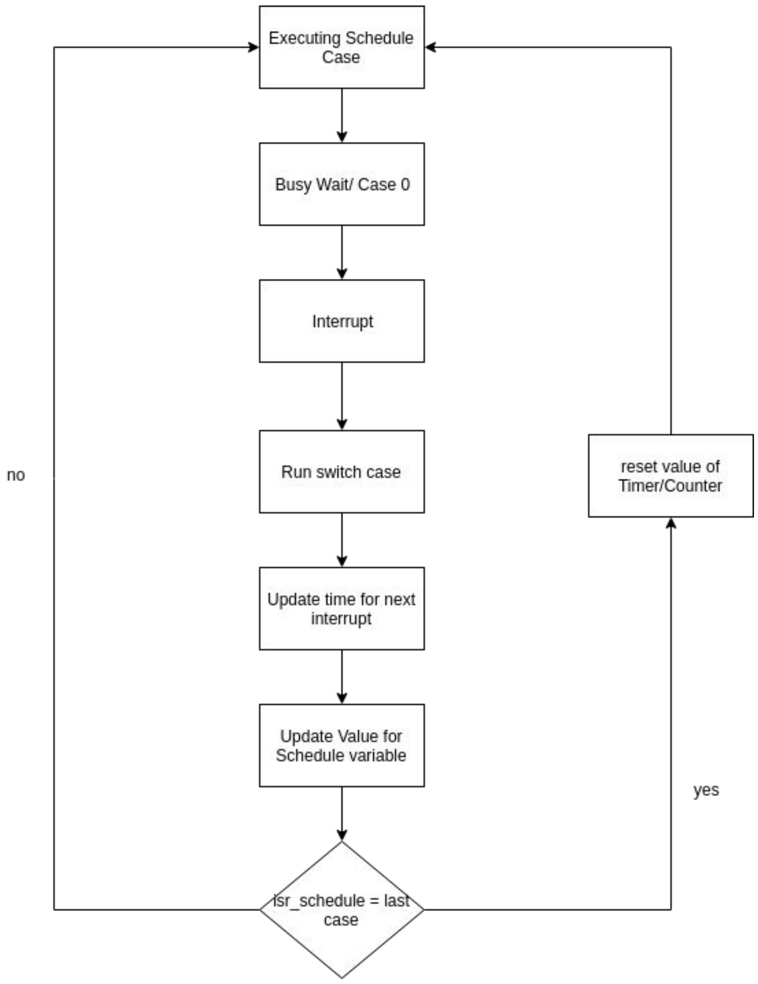
Equivalent Schedule

Schedule Breakup

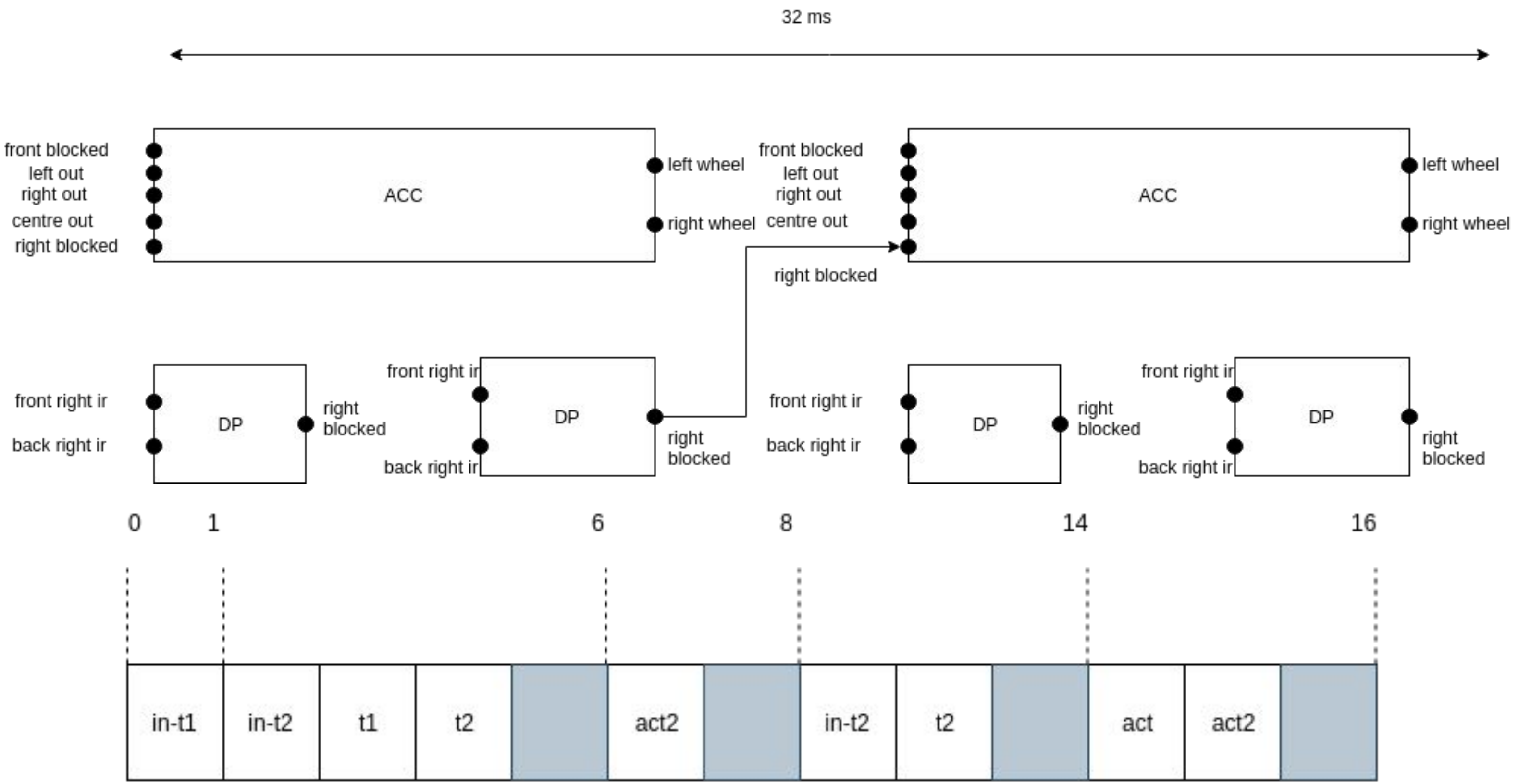


Phase 3 COmpiler





Case Study-Acc+detect parking



CONCLUSION

- We have successfully demonstrated the applicability of our TTProg[Heptagon] High Level Programming technique to robotics
- Architecture and processor independent technique
- WCET needs to be provided
- Static Scheduling
- Highly efficient code
- Deterministic Dataflow
- No context switching,memory allocation and RTOS required

Future WOrk

- Multiple modes
- Error handling in compiler
- Option to choose different processor prescaler
- Extend to multi processor systems
- Automatic technique to decompose large task to small task
- Needs to be applied to complex robot programming

References

I. Thomas A. Henzinger, Benjamin Horowitz and Christoph M. Kirsch,"Giotto: A Time-triggered Language for Embedded Programming", in Proceedings of the IEEE, vol. 91. IEEE, 2003

II. David Harel, ”Statecharts: A visual formalism for complex systems”, in Science of Programming, vol. 8. Elsevier, 1987.

III. George Wells, "Coordination Languages: Back to the Future with Linda" in Proceedings of WCAT’05. 2005

IV. Pascal Raymond and Nicolas Halbwachs, "The Lustre Language". URL: http://www-verimag.imag.fr/~raymond/edu/eng/lustre-a.pdf

V. Heptagon/BZR manual. 2017. URL : \url{http://heptagon.gforge.inria.fr/pub/heptagon-manual.pdf}