**RTOS SCHEDULER MODULE**

**Implementation**

We implemented a Binary Progression Scheduler (BPS) that manages the access to the CPU recourses in a controlled way.

When there is not a Systick interrupt the **SchM\_Background** function is executed

**SchM\_OsTick**

**SchM\_OsTick**

**SchM\_OsTick**

**SchM\_OsTick**

SysTick 781.25 micro-seconds

**SchM\_OsTick**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

void **SchM\_OsTick**( void )

{

//Counter (int) to tasks´ id

//Ostick counter increments

//Set Ready the Task if mask match counter

for(… ){ // Loop controlled by the number of tasks

if( ){ // if((Counter & Mask) == Offset)

// set the Task´s state in READY state

}

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SchM\_Background**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

void **SchM\_Background**( void )

{

//Counter (int) to tasks´ id

for(;;) {// Infinite loop

for(… ){ // Loop controlled by the number of tasks

if( ){ // if a Task is in READY state

//set the Task´s state in RUNNING state

//set the scheduler´s state in RUNNING state

//call the task (callback function)

//set the Task´s state in SUSPENDED state

//set the scheduler´s state in IDLE state

}

}

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

We added 2 bit flags to control and indicate when an overload occurs

**SchM\_OsTick**

**SchM\_OsTick**

**SchM\_OsTick**

**SchM\_OsTick**

**T1**

((Counter & Maskt1) == Offsett1)

**….**

**T2**

((Counter & Maskt2) == Offsett2)

**OVERLOAD**

typedef struct{

uint8\_t FlagOverLoad: 1; //Flag for indicating that an Overload have occured

uint8\_t FlagTaskReady:1; //Flag for indicating that there is a task in Ready/Running state

} Flags;

The system turns on a LED when overload occurs

**SchM\_OsTick** whit overload

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

void **SchM\_OsTick**( void )

{

//Counter (int) to tasks´ id

//Ostick counter increments

//Set Ready the Task if mask match counter

for(… ){ // Loop controlled by the number of tasks

if( ){ // if((Counter & Mask) == Offset)

if(){ //Review if there is any Task in Running or Start State before activating other task (FlagTaskReady)

//Set the OverLoad Flag

//TurnOn the OVERLOADPIN -> TurnOnOverloadPin();

}

// set the Task´s state in READY state

//Set the Flag that indicates the activation of a Task (FlagTaskReady=1)

}

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SchM\_Background** whit overload

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

void **SchM\_Background**( void )

{

//Counter (int) to tasks´ id

for(;;) {// Infinite loop

for(… ){ // Loop controlled by the number of tasks

if( ){ // if a Task is in READY state

//set the Task´s state in RUNNING state

//set the scheduler´s state in RUNNING state

//call the task (callback function)

//set the Task´s state in SUSPENDED state

//Clear the Flag that indicates that there is one Task Activated (FlagTaskReady=0)

//set the scheduler´s state in IDLE state

}

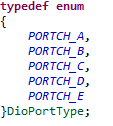
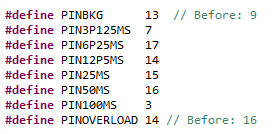
}

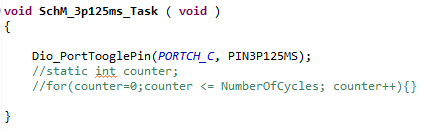
}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Testing the system (Tasks)**





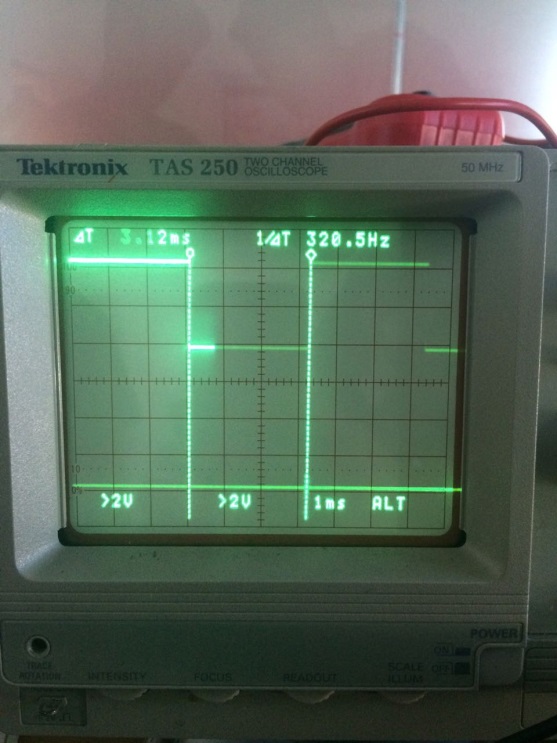


Ilustración 1. Task 3.125 ms

.

T

3.125 ms

3.125 ms

3.125 ms

3.125 ms

3.125 ms

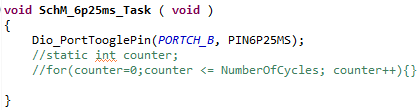




Ilustración 2. Task 6.25 ms

.

T

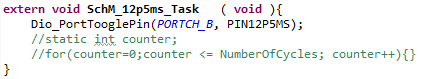
6.25 ms

6.25 ms

6.25 ms

6.25 ms

6.25 ms



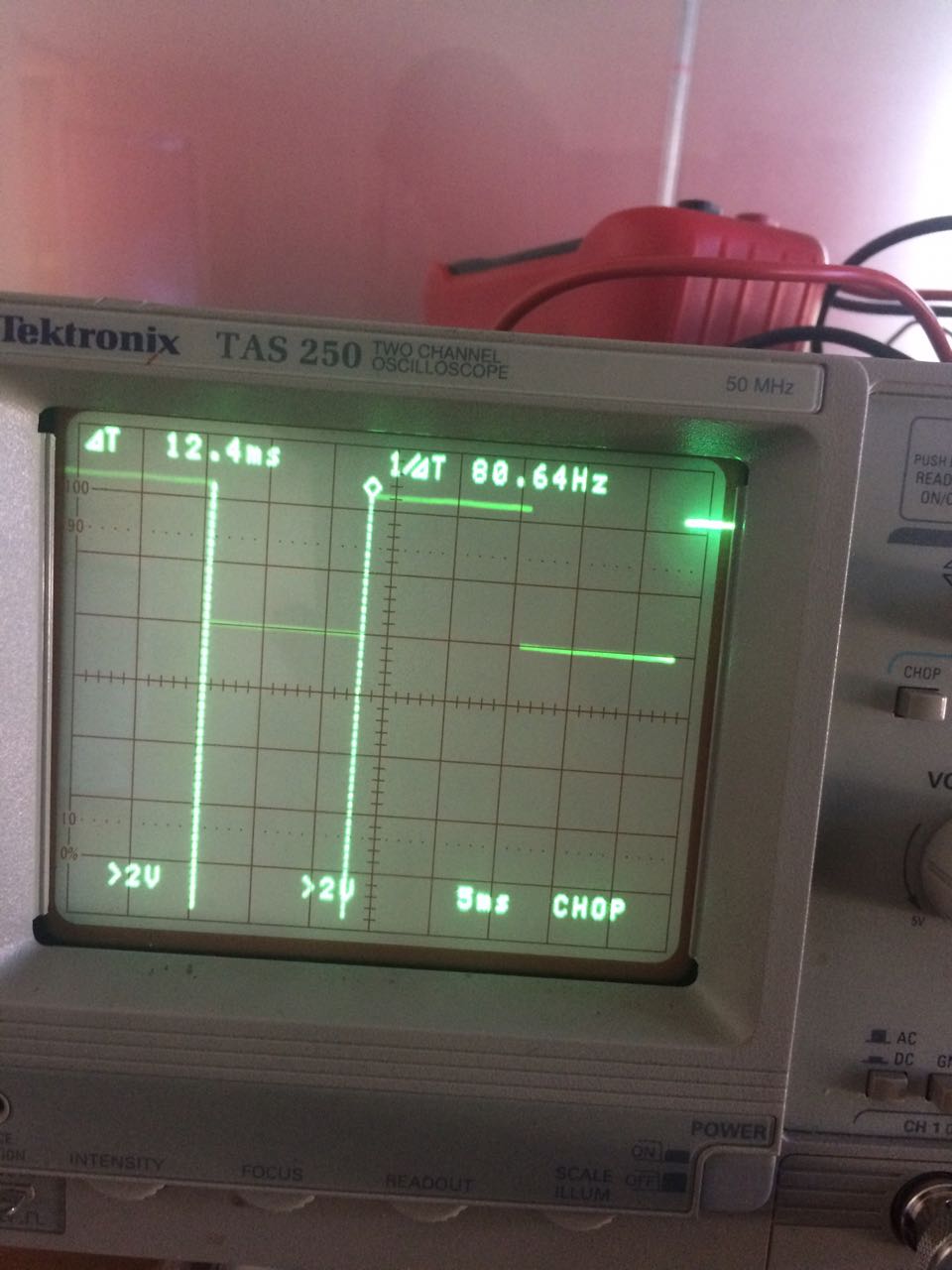


Ilustración 3. Task 12.5 ms

.

T

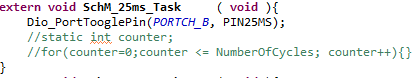
12.5 ms

12.5 ms

12.5 ms

12.5 ms

12.5 ms



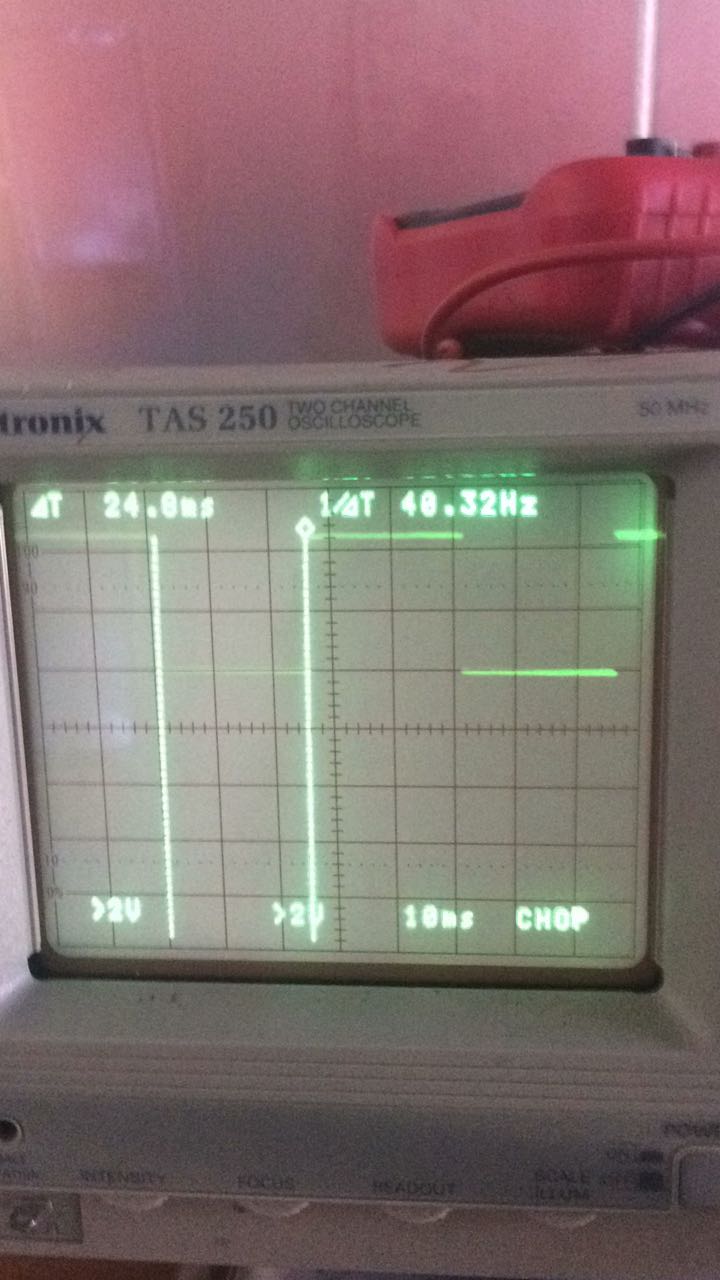


Ilustración 4. Task 25 ms

.

T

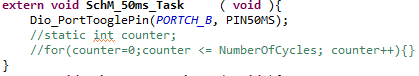
25 ms

25 ms

25 ms

25 ms

25 ms



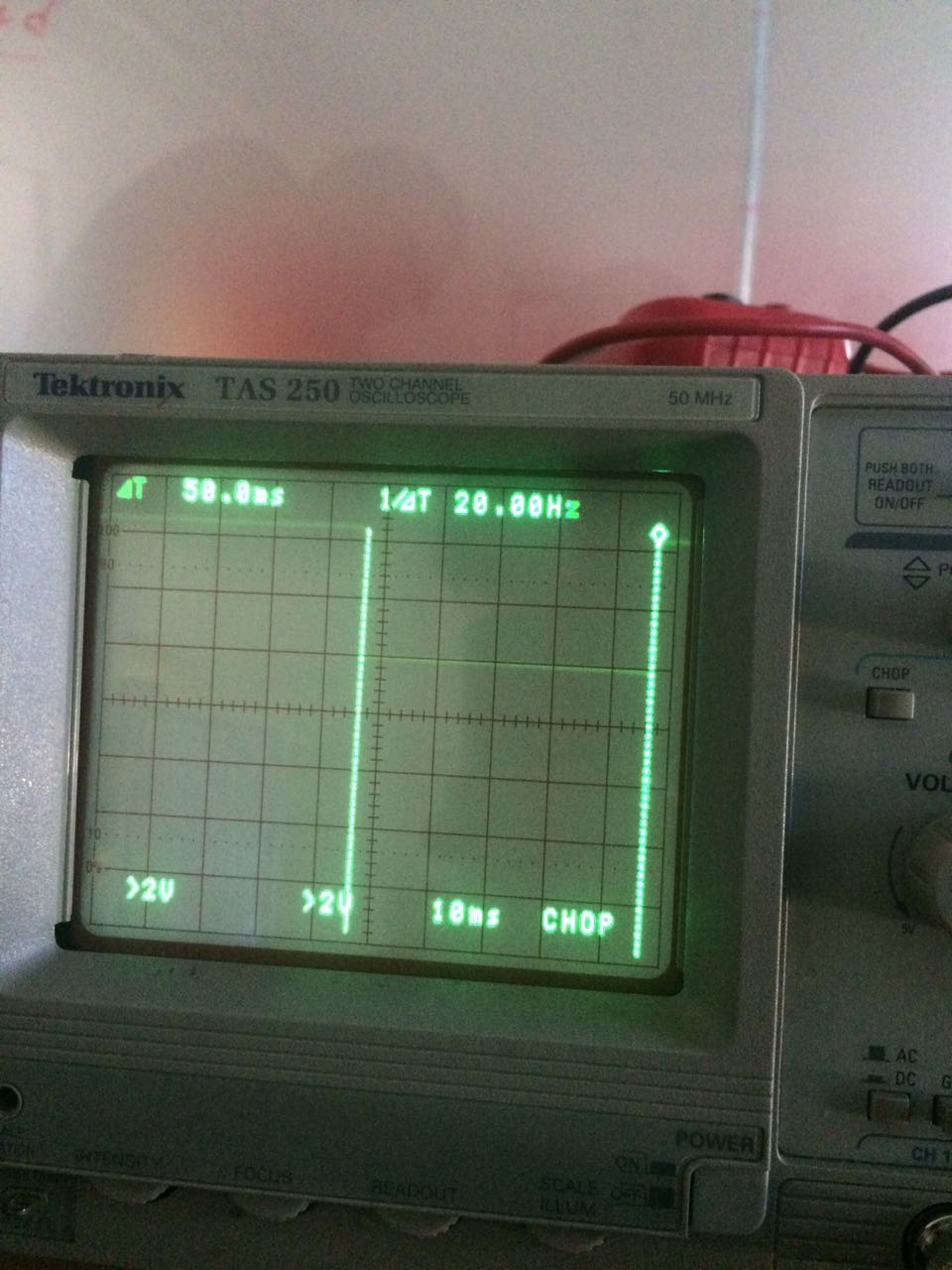


Ilustración 5. Task 50 ms

.

T

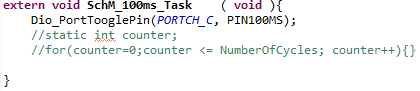
50 ms

50 ms

50 ms

50 ms

50 ms



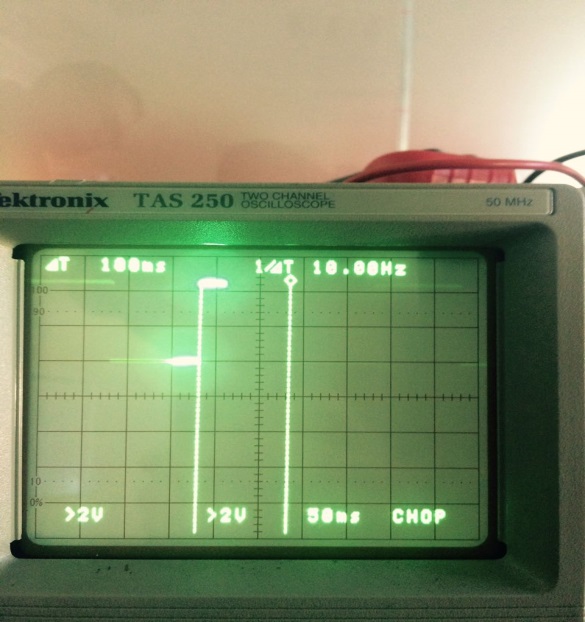


Ilustración 6. Task 100 ms

.

T

100 ms

100 ms

100 ms

100 ms

100 ms

**Testing the system (CPU)**

**void** **SchM\_<TaskPrefix>\_Task** ( **void** )

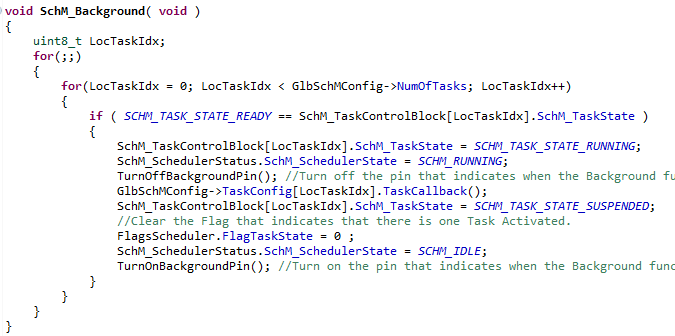
{

Dio\_PortTooglePin(*PORT*, PIN);

**static** **int** counter;

**for**(counter=0;counter <= NumberOfCycles; counter++){}

}



For this analysis we didn´t consider the instruction of **Toogle Dio\_PortTooglePin(PORT,PIN)**

For this analysis when NumberOfCycles gets 10000 the time of the Task´s execution is 2800 microseconds.

When NumberOfCycles is configured whit 2700 the approximated time of the Task´s execution is 756 micros.

When NumberOfCycles is configured whit 0 the approximated time of the Task´s execution is 0 micros.

**Testing the system (OVERLOAD)**

**void** **SchM\_<TaskPrefix>\_Task** ( **void** )

{

Dio\_PortTooglePin(*PORT*, PIN);

**static** **int** counter;

**for**(counter=0;counter <= NumberOfCycles; counter++){}

}



For this analysis we didn´t consider the instruction of **Toogle Dio\_PortTooglePin(PORT,PIN)**

For this analysis when NumberOfCycles gets 10000 the time of the Task´s execution is 2800 micro seconds.

When NumberOfCycles is configured whit 3000 the approximated time of the Task´s execution is 840 micro seconds .

**SchM\_OsTick**

**SchM\_OsTick**

**SchM\_OsTick**

**SchM\_OsTick**

**T1**

((Counter & Maskt1) == Offsett1)

**….**

**T2**

((Counter & Maskt2) == Offsett2)

**OVERLOAD**